AD-A243 687

DTIC
ELECTE
DEC 2 7 1991
S D
C

AFIT/GCS/ENG/91D-17

Evaluation of Scalar Value Estimation Techniques For 3D Medical Imaging

THESIS

Rob W. Parrott
Captain, USAF

AFIT/GCS/ENG/91D-17

Approved for public release; distribution unlimited

91-19010

91 12 24 006

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Master's Thesis

4. TITLE AND SUBTITLE

Evaluation of Scalar Value Estimation Techniques For 3D Medical Imaging

5. FUNDING NUMBERS

6. AUTHOR(S)

Rob W. Parrott, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/91D-17

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Scalar value estimation in 3D medical imaging increases data resolution for enhanced renditions and corrects inaccurate surface formations. Accurate estimations are vital because clinical assessment is often aided by examination of 3D medical images. This thesis explores different estimation techniques and introduces the geostatistical estimation technique called kriging to the field of 3D imaging. Kriging theory claims to be the optimal estimator – better than the standard deterministic methods commonly used. The techniques investigated are linear interpolation, trilinear interpolation, tricubic interpolation, and kriging. This research investigates scalar value estimation in the volume pre-processing operation of slice interpolation and in a surface extraction method called cell subdivision. Tricubic interpolation is shown to be most useful in artificially created volumes of smooth functions. It is also shown to produce poor results in medical volumes and in slice interpolation. More importantly, this research demonstrates that kriging subsumes the deterministic methods investigated and can estimate much better than tricubic interpolation.

14. SUBJECT TERMS

COMPUTER GRAPHICS, MEDICAL COMPUTER APPLICATIONS, INTERPOLATION, STATISTICS, MEDICAL IMAGING, KRIGING

15. NUMBER OF PAGES

173

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

AFIT/GCS/ENG/91D-17

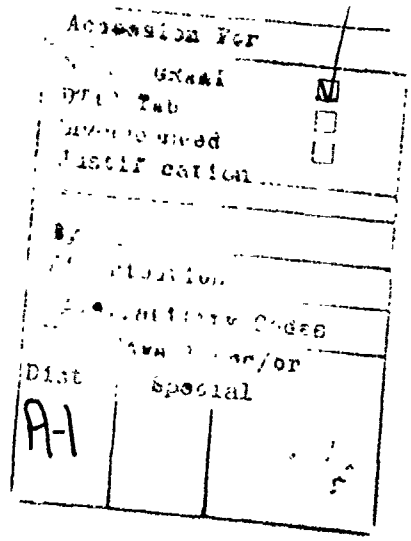
Evaluation of Scalar Value Estimation Techniques For 3D Medical Imaging

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Rob W. Parrott, B.S. in Computer Science
Captain, USAF

December, 1991



Approved for public release; distribution unlimited

Table of Contents

List of Figures

List of Tables

Acknowledgments

My first and ultimate thanks go to my Lord and Saviour, Jesus Christ. Through Him I found the strength and confidence to keep going when I thought I wasn't capable. I thank God, His Holy Spirit, and His Son for being with me and my family. With Them in our lives and Their eternal perspective in our hearts, we grew stronger in our faith during this time. Thank you Lord for making this a learning and at times an enjoyable experience for me and one that I will cherish for the rest of my life.

Special thanks go to my family. Thank you Verla for supporting me through all the times I wasn't with you, especially weekends (I know, what's a weekend?). I'm very grateful for your strength and patience. Misty, Alex, and Wesley - I'm sorry you had a part-time Dad for the last 18 months, but I'm back full-time and we're a family once again!

I would like to thank my advisor, Col Martin Stytz, who gave me invaluable support and guidance in the area of 3D medical imaging and technical writing. His previous papers and dissertation greatly aided in my understanding of 3D medical imaging. Thank you Col Stytz for your constant encouragement and positive outlook. These qualities helped keep me motivated throughout the research.

Thanks also go to Col Phil Amburn and Maj David Robinson, who both provided me with crucial technical guidance. Thank you Col Amburn for first recommending the use of marching cubes and kriging in this research. Thank you also for your help with GPR, tricubic interpolation, and numerous other areas where your keen insight pushed me in the right direction. Thank you Maj Robinson for your assistance in the area of kriging. With Your deep understanding of this complex estimation technique, you helped me a great deal to see how it could be applied in 3D imaging.

Thanks go to Capt Chris Brodkin for sharing his kriging code with me. Without this, my work would have been much more difficult.

Finally, I'd like to thank the fellow students who listened to me when I needed feedback. This act of kindness was a tremendous aid when code was seemingly debug-proof or concepts just didn't seem clear. Thanks for your comradeship - Pat Rizzuto, Patty Brightbill, Don Duckett, and Wayne Mcgee.

Rob W. Parrott

Abstract

Estimation has not received enough attention in 3D medical imaging. Estimation is often done in 3D medical imaging to increase data resolution for enhanced renditions. It is also used for correcting inaccurate surface formations in the well known marching cubes algorithm. Accurate estimations are vital because clinical assessment is often aided by examination of 3D medical images. This thesis introduces the geo-statistical estimation technique called kriging to the field of 3D imaging. Kriging theory claims to be the optimal estimator - better than the standard deterministic methods commonly used.

This thesis explores four estimation techniques for use in 3D medical imaging. The techniques are linear interpolation, trilinear interpolation, tricubic interpolation, and kriging. The interpolation methods are standard estimation techniques used in 3D imaging. The estimation techniques are used to estimate scalar values in two primary areas. These are intra-cell scalar value estimation and the volume pre-processing operation of slice interpolation. This research investigates intra-cell scalar value estimation in a surface extraction method called cell subdivision. This research also explores slice interpolation by estimating scalar values between existing medical data slices. Slice interpolation is the operation of estimating logical slices between existing ones, typically to increase data resolution to obtain a finer mesh representation of a surface.

Tricubic interpolation is shown to be most useful in artificially created volumes of smooth functions. It is also shown to produce poor results in medical volumes and in slice interpolation. More importantly, this research demonstrates that kriging subsumes the deterministic methods investigated and can estimate much better than tricubic interpolation.

Evaluation of Scalar Value Estimation Techniques For 3D Medical Imaging

I. Introduction

A twelve-year-old boy suffers from severe congenital defects in the sacrum, lumbar spine, and acetabula (hip). The surgeon assigned to this case reviews both two-dimensional (2D) and three-dimensional (3D) medical images of the boys pelvic region. The 3D image is generated by a computer graphics program with input from a series of 2D Computed Tomography slices (scans) of the patient's pelvic region. 3D medical images provide views of internal organs, bones, tissues, muscles and other body parts otherwise only seen after the human body has been invaded by exploratory or therapeutic surgery. Because of the additional insight revealed by the 3D images, the surgeon decides not to perform surgery (45). In this case, 2D scans alone do not provide enough information for the surgeon to make an accurate assessment of the patient's condition. Analysis of craniofacial abnormalities, radiation treatment planning for cancer patients, analysis of pelvic deformities , and cardiopulmonary analysis are only a few of the areas 3D medical images have assisted medical clinicians make vital decisions in patient management (45).

Because of the need to visualize volume (3D) data sets for analysis, computer scientists have developed many different techniques for portraying this data. Methods exist to depict this three-dimensional data in 3D; however, the emphasis has been to display the 3D data on a 2D screen (26) and (27). This research explores a subset of the latter methods in the area of medical imaging, with an emphasis on estimation of scalar values.

To understand the exact problem considered and the purpose of this thesis some background information must be presented first. Following the background, the problem is discussed. This chapter ends by presenting the purpose of the thesis followed by the research goals.

1.1 Background

Three dimensional data visualization encompasses all methods that render images from volume data sets. Medical imaging and scientific visualization are two broad application areas under 3D data visualization. The need for accurate representations of the physical volume separates medical imaging and many scientific visualization areas from other 3D computer graphics techniques. The latter techniques mostly attempt "... to form realistic images from scene descriptions which may, or may not, have a physical counterpart" (41:2). Accuracy in medical images is important because many clinical applications rely on the faithful representation of the portions of the human body scanned and reproduced three-dimensionally.

3D medical imaging is composed of three high level processes : "data collection, 3D data display, and data analysis" (41:2). This work is concerned with 3D data display – a computer graphics task. Data collection and data analysis are both performed by end-user experts – radiologists and clinicians.

Data collection is defined by Stytz (41:2) as

> ... the radiologist's arena, and deals with issues concerning the statistical significance of collected data, patient dosage, medical imaging modality operation, development of new techniques/modalities for gathering data, and 2D image reconstruction.

The main data collection scanning technologies (or modalities) include Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Single Photon Emission Computerized Tomography (SPECT), and Positron Emission Tomography (PET) (41) and (39). The main purpose of each of these modalities is to sample data in patient space and produce a series of 2D images corresponding to some aspect of the patient such as soft and bony tissue in the case of CT and MRI studies and metabolic measurements in the case of PET and SPECT studies. CT and MRI are used primarily for imaging anatomical structures, whereas SPECT and PET are used mainly for biochemical imaging. For each modality, different properties of the human body are imaged, but all produce a series of gray scale 2D images. Gray scale means values range from 0 to 255 (8 bits per value). For the purposes of this thesis, it is enough to say the values correspond to some material property of the imaged volume, such as density.

CT was the first modality, following the advent of X-ray technology, to provide high quality 2D non-invasive images. The others followed soon thereafter to provide images of equal (if not better) quality. Since this research deals with 3D data display, reference Stytz for an overview of how each of these modalities are constructed and how their operation influences image quality. (39).

These 2D images alone provide a certain level of non-invasive assistance, but with these images the clinician must mentally build 3D objects of interest. This may be very difficult, depending on the experience of the analyzer and the application area. In one area of clinical study, according to Wojcik and Harris (45:197),

> Mental integration of all of the images frequently obtained to evaluate a complex skeletal problem (i.e., plain radiographs, polydirectional tomography, and MPCT [multiplanar CT]) is frequently difficult even for experienced radiologists and is even more difficult for less experienced radiologists or nonradiologists [such as surgeons].

A four year study that assessed 3D imaging from CT scans at the Department of Diagnostic Radiology, University of Manchester concluded that "3D imaging does have a useful role to play in a number of specific clinical situations when used in conjunction with CT and other radiological imaging methods" (45:103-144).

The main challenge faced by 3D data display graphics researchers is how to create clinically useful accurate 3D images quickly from the immense amount of information obtained from these imaging modalities. One modality can produce up to 35 mega-bytes of data for a single patient (41). This occurs because a typical study consists of 12 to 100 or more slices (scans) of images with resolutions up to 512 X 512. Following the development of CT in the early 1970's, many software algorithms were created to produce 3D images from CT data. The methods conceptually combine the 2D CT data into a volume of information from which significant data is portrayed. Most other 3D visualization methods (such as scientists and engineers studying computation fluid dynamics, molecular modelling or the earth sciences (36)) have their roots in early medical imaging techniques (45:60). These methods generally fall into two main categories - volume and surface methods.

Most 3D imaging researchers use the terms volume rendering and surface rendering to classify rendering of volume (3D) data sets. Surface rendering methods

extract a surface or surface boundary from the volume data set and represent it using some data structure. Volume rendering directly processes voxels (volume elements), assuming that each voxel is either opaque or partially opaque. This classification can be misleading in some cases. because it implies that surface rendering only renders surfaces and volume renderers only render volumes. Yet, surface renderers can render volumes (multi-surfaces using opaque and transparent surfaces) as well as just single surfaces. Also, volume renderers can display distinct surfaces much like surface renderers by processing binary volumes (45). A binary volume is formed by processing only those voxels forming part of a surface of interest. Then only the the selected voxels are rendered. It is termed binary because voxels are classified as contributing to a structure of interest (assigned a 1) or not (assigned a 0).

I prefer Farrell's terminology (19) of surface unit based and volume unit based approaches, that differentiates not by what the final image consists of, but rather by the type of unit or "primitive image element" used in the visualization process. For example, two-dimensional primitive image units, such as planar polygons or voxel faces, are used in the surface unit based approaches, whereas volume elements (voxels) is one type of three-dimensional primitive image unit used in volume unit based approaches (see chapter two for a more complete discussion of these approaches). Hereafter in this document. the terminology *surface method* and *volume method* corresponds to surface unit based and volume unit based approaches. respectively.

Surface and volume methods have their supporters in both the scientific and medical communities. The choice of method depends on the application and often on the personal preference of the researcher. scientist, engineer, radiologist. or clinician. Many researchers (.e.g (47),(43), and (27)) have noted that a variety of approaches used to render the same data set provide crucial clues to analyzing and understanding the data, whereas only one approach might leave gaps in knowledge. In the case of medical imaging. surface methods are good for displaying surfaces with definite boundaries. such as bony tissue in the human body. Likewise, volume approaches are superior to surface approaches for depicting *fuzzy* or amorphous volumes such as diffused tumors and blood flow in arteries and veins (33), (16) and (45). Also, surface methods allow for one of the most useful medical imaging applications - interactive manipulation of structures (45). One purpose of this task is surgery rehearsal. Surgery rehearsal is not very practical with images rendered via volume methods because they typically are too slow for interaction. Surface methods allow

real-time manipulation because they normally produce geometric primitives such as planar polygons. These geometric primitives can be used as input into fast hardware or software based hidden surface removal and shading algorithms. However, if a new surface is desired for rendering, the entire volume must be re-processed again. Volume methods require a high computational cost for processing all the data elements in the volume; however, the increased cost may be less important than the need for a diffused or fuzzy rendition of the data.

The third medical imaging process data analysis, involves obtaining "quantitative information about the structure in the scene" (45:57). Quantitative information includes the average density of an area within the 3D image, the size of certain anatomical structures such as bones and blood vessels, or the volume of a sub-region. Much of the desired analytic information can be acquired directly from the 2D slices. Measurements required from 3D images are volumes, 3D distances, 3D angle measurements, and "other less commonly used measurements... [such as] center of mass, moment of inertia, and surface curvature." Surface methods as well as certain volume methods (which allow the concept of a structure) provide the ability to perform all the 3D data analysis operations (45:58).

Image accuracy is critical to the data analysis process. Inaccuracies in an image can adversely affect these measurements, possibly leading to erroneous assessments by the clinician.

1.2 Problem

Scalar value estimation is very common in many 3D imaging algorithms; however, little has been done to investigate different scalar value estimation techniques.

The level of inaccuracy has not been deemed serious because cost has been a larger concern. As Herman and Liu noted in 1979, "... reduction in cost is essential; computer time for the display will have to be borne by the patient." Cost is certainly still an issue (45:224). Higher order deterministic functions such as quadratic or cubic polynomials and statistical based estimation methods are more computationally expensive than simple linear interpolation. Yet, the results obtained from using them can greatly improve image accuracy and fidelity – thus improving clinical assessment and quantitative analysis. Since workstations are becoming more powerful, this issue of cost will become less of a problem; hence, a search for more accurate estimation methods is necessary.

Accuracy in medical imaging is very important to the medical community. Wojcik and Harris note, "the primary purpose of radiologists is to provide the most accurate diagnostic information possible within the capabilities of available imaging modalities" (45:196). This estimation problem may cause serious 3D image errors in two primary areas:

- The volume pre-processing operation of slice interpolation.
- Intra-cell scalar value estimation[1].

Udupa and Herman state, "the interpolation problem in our opinion, has received less attention than it deserves" (45:13). The interpolation problem they are referring to is the process of determining new slices between existing image slices to form a cube shaped volume. Udupa and Herman discuss the primary methods of interpolating new slices are nearest neighbor,linear interpolation of voxel values, and trilinear interpolation. The nearest neighbor approach produces the worst results of the three because it does no estimation. This approach simply assigns new voxel values to the interpolated slices to the nearest original voxel. Trilinear performs somewhat better than linear, but the variation in both cases is still assumed to be linear when in fact it may not be. Udupa and Herman investigated one other method they created for use in binary volumes. This method is termed shape based interpolation. Voxels are mapped from the binary volume to a second corresponding array that holds distances from each voxel to the boundary. If the voxel in the binary volume was 1, the distance in the second corresponding array will be positive, else negative. Linear, trilinear, or some other interpolation scheme is used to derive new distances in the second array. The newly interpolated distances are then mapped to a new binary volume consisting of new slices between existing ones. Positive distances are assigned a 1 in the binary volume, negative distances a 0. Hence, the boundary of the structure of interest between 0 and 1 entries in the binary volume influences the interpolation. The authors claim it provides more accurate quantitative analysis and in their opinion leads to a better surface representation in surface methods. However, this method will not work in volume methods unless the volume method uses a binary volume.

[1] A cell is a logical cube with vertices formed by four voxels in one data slice and four voxels in an adjacent slice.

Many 3D rendering algorithms (primarily surface methods) require cubic shaped voxels (28), (44), (22), and (45). Voxels are volume elements, the 3D analog of pixels. Cube shaped voxels are termed cuberilles. Other rendering algorithms, including volume methods such as (33:34) , interpolate new slices to improve image quality. For example, assume a study consists of a series of 100 256 pixel X 256 pixel CT images. To make this volume cubic in shape, new slices are created in between existing ones until there are 256 total slices. The extra 156 slices are usually determined by linear interpolation of the sampled CT values between the original slices. The problem with this method is the data may not vary linearly.

Intra-cell scalar values are estimated to improve image fidelity and correct possibly inaccurate surface renditions generated by cell interpolation surface methods[2]. Wilhelms and Gelder (50) have shown that the commonly used trilinear interpolation estimation method does not estimate values in artificially created volumes (versus scanner generated) as accurately as a parametric cubic function. They demonstrated that tricubic interpolation produces better images than those produced by trilinear interpolation.

The problems with linear, trilinear, and tricubic interpolations are that these methods assume the variability of the data and assume a neighborhood of sample values that influence the estimation. However, variation is not necessarily linear or cubic in nature and the number of sample values that should influence the estimation can be different than the unchangeable number determined by these methods. Assumptions about the variation of the data and the neighborhood of sample values can possibly produce erroneous results in the estimation process. Instead of assuming a variation, a geo-statistical process exists that can aid in determining the variation of the data for the purpose of estimating new values. In this same process, the number of sample values influencing the estimation can be modified to fit the variability of the data. This process is termed kriging.

1.3 Purpose

The purpose of this research is to investigate the application of several estimation techniques to estimating scalar values within computational cells and during the volume pre-processing operation of slice interpolation.

[2]Cell interpolation surface methods are described in chapter two.

To accomplish the purpose of this thesis, four estimation techniques were implemented. These are linear interpolation, trilinear interpolation, tricubic interpolation, and kriging. This is the first time the geo-statistical estimation technique called kriging has been applied in 3D medical imaging.

Kriging differs from the deterministic methods implemented[3] because it is a statistical method. That is, kriging accounts for error. Not only is it a statistical estimation multiple linear regression model, but it is a geo-statistical model as well. A geo-statistic differs from a classical statistic by how the variables are treated in the estimation process. A classical statistic views data as random. A geo-statistic accounts for spatial inter-dependence of the variables used in the estimation. Kriging has proven to be very successful in estimation applications in the geo-sciences. Since medical data is spatially distributed and has regions in which sample values are highly correlated (e.g., bone and tissue regions), this technique is applicable to medical data.

Kriging theory shows that kriging is the optimal estimator. It is optimal in a statistical sense in that it minimizes estimation error variance and removes bias in the estimation. This research demonstrates that this optimal estimator subsumes the deterministic methods investigated and in some cases produces visually better results.

1.4 Approach

The main goal of this research is to compare the results of different techniques for estimating scalar values. Estimated values are used in two areas – within computational cells and for creating logical slices during slice interpolation. To accomplish this goal, the following tasks were accomplished :

- A cell interpolation algorithm was implemented, including the capability of subdividing cells for intra-cell scalar value estimation.
- Trilinear interpolation, tricubic interpolation, and kriging estimation functions were implemented for comparison.
- Artificial volume data sets were built for comparing and contrasting methods.

[3]The deterministic methods are linear, trilinear, and tricubic interpolation. Determinism means they do not account for any variation such as systematic error in sampling.

- A rendering system was developed by modifying an existing one (to support software engineering goal of re-usability) to display images for comparison.
- 3D Images were generated by extracting surfaces from both artificial volumes and actual medical data sets by using trilinear interpolation, tricubic interpolation and kriging estimation techniques.
- 2D Images were generated by estimating scalar values between two medical image slices using linear interpolation, tricubic interpolation, and kriging estimation techniques.
- Values and images derived from the different estimation techniques were compared.

1.5 Overview

The remainder of this document consists of the following chapters. Chapter two provides an introduction to 3D medical imaging methods, focusing mainly on surface methods. Chapter three describes kriging theory. Chapter four provides implementation details of the algorithms developed during this effort. Chapter five contains the images resulting from the research and a discussion of these results. Chapter six consists of a section recommending future research following this effort and a section with concluding remarks. The appendices provide further implementation details for the interested reader.

II. 3D Medical Imaging

This chapter discusses 3D medical imaging concepts, concentrating on the areas most pertinent to this research. In particular, 3D data representation and 3D medical imaging transformations are summarized, followed by a discussion of the major 3D medical imaging methods – contour, surface and volume. Because they were used in this research, surface methods are discussed in more detail than the other two areas. More comprehensive backgrounds of 3D medical imaging concepts can be found in (45), (19), and (41).

I chose to implement a cell interpolation surface method – based on the marching cubes algorithm (34) (described later in this chapter) – because the subdivision technique developed by Wilhelms and Gelder (50) provided me with an excellent framework to explore kriging in the intra-cell estimation process.

Inaccuracies occur in the marching cubes algorithm because it can incorrectly define the surface of interest in a volume. The algorithm assumes the original sampled data is sufficient to represent the surface accurately, when in fact it may not be. A correct rendering of the surface never occurs because surfaces are continuous and we are sampling from this continuous domain. The best alternative would be to sample finer, but with current scanning technology limitations, costs, and patient health, this is not always possible. Because obtaining a finer sampling can be difficult and costly, Lorensen and Cline (8) and Wilhelms and Gelder (50) obtain more accurate renditions by subdividing portions of the volume and estimating new scalar values within the subdivided volume areas, given a surrounding neighborhood of original sampled values.

Wilhelms and Gelder (50) investigated two estimation methods – trilinear and tricubic interpolation. In this thesis effort, I explore a statistical estimation function derived by a process called kriging. Kriging, according to Matheron, "is the probabilistic process of obtaining the best linear unbiased estimator of an unknown variable" (25). If properly applied, kriging, in contrast to trilinear and tricubic, analyzes the data prior to estimation to determine the actual variation of the data and minimizes the estimation error variance. See chapter three for a detailed discussion of kriging.

2.1 3D Data Representation

To understand how medical images are produced, one must first become familiar with some of the basic concepts of 3D imaging from 3D data sets.

Most algorithms developed for rendering 3D data sets expect the data to be regular shaped, or in a 3D lattice or grid format. A lattice or grid in this context is best visualized as a framework of parallel planes in space, with data information (such as material density) conceptually located at the intersection of the planes or within the volume of space between planes. In many cases transformations are applied to the discretized data to remove sampling noise, to alter the resolution of the data, or to make an irregular data set regular (45:4-14).

3D medical data is regular because it is derived from conceptually stacking 2D same resolution scans, e.g., from CT or MRI output. The third dimension is then realized as the stack or slice number (see figure 2.1). Regular shaped data is popular

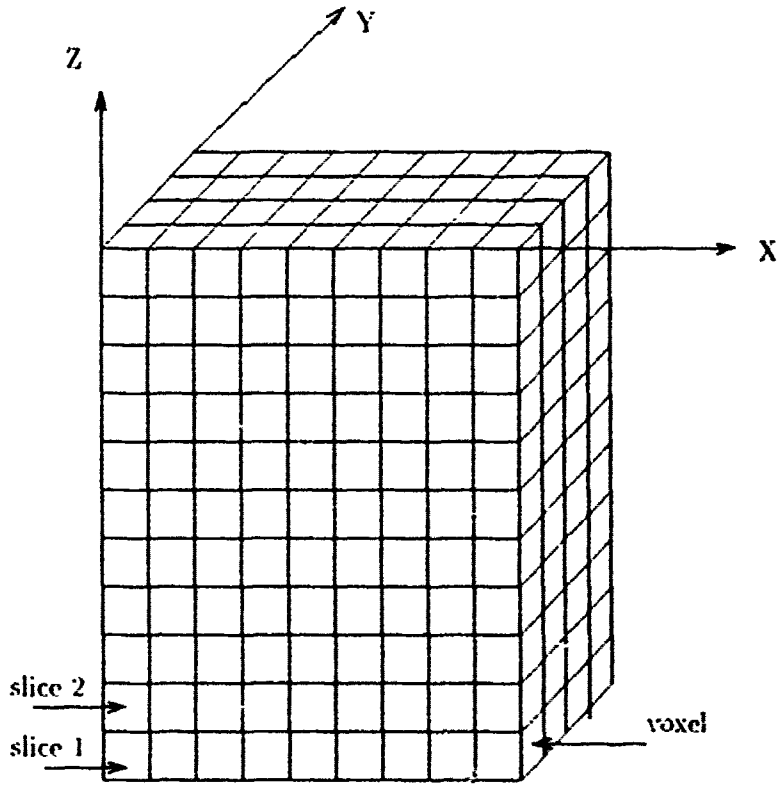


Figure 2.1. Pictorial representation of a medical data volume

because it can be easily mapped directly into a 3D array. Once in the array, data

is ordered for manipulation. One method of ordering is simply to view elements in the array as volume elements, or voxels. Voxels are conceptually the equal-sized parallelepipeds created by the intersections of three sets of parallel planes, each set orthogonal to the other two. The voxel and cuberille data models have been widely used in imaging agorithms. The use of these data models is discussed in the section titled 3D Medical Imaging Methods.

2.2 3D Medical Imaging Transformations

The 3D medical imaging transformations can be seen in figure figure 2.2, adapted from (45). These are operations upon the data that transform it from scanner output to a 3D graphics image projected onto a 2D screen. The medical image data, input into scene space, are the 2D image slices output from scanners. One scene transformation from scene space to scene space processes the input data by filtering it to suppress noise or to determine tissue boundaries. Forming new slices by estimating values between existing images is a common scene transformation (also called slice interpolation). Another scene transformation is a Volume Of Interest (VOI) operation. A VOI operation pre-processes the scene data to include only the data that contributes to a structure of interest. A structure of interest might be a single organ or a portion of a volume, like the left hemisphere of a brain study. Structure extraction is the step in the 3D medical imaging pipeline that generates some structure, such as a polygon mesh, from scene space into object space. A VOI operation only encloses the structure of interest in scene space whereas structure extraction generates a separate data representation of the structure of interest. Structures in object space are possibly transformed by geometric transformations (rotations, translations, or scalings) into image space. Rendering (projective transformations) transforms the image space structures into view space - the 2D computer graphics screen. The final data transformation, analysis. acts on the 2D image to provide parameters for analytic measurements, discussed as data analysis in chapter one.

2.3 3D Medical Imaging Methods

3D medical imaging methods are typically divided into three main approaches: contour, surface and volume. The techniques differ in the dimensionality of the geometric data model used to create the image - 1D contours (1D units) for the

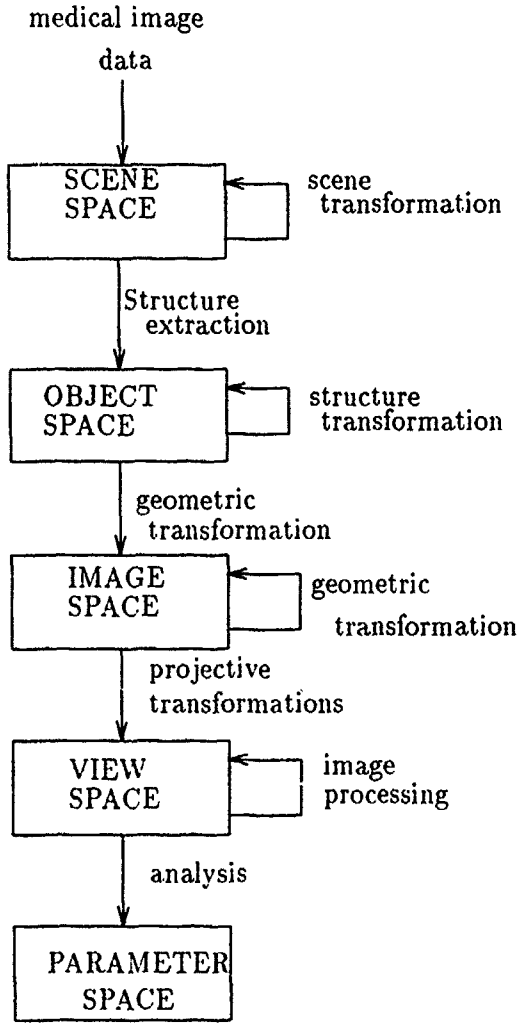


Figure 2.2. Pictorial Representation of 3D Medical Imaging Transformations

contour approaches, 2D polygons (2D units) for the surface methods, and voxels and computational cells (3D units) for the volume methods (See figure 2.3). Only those

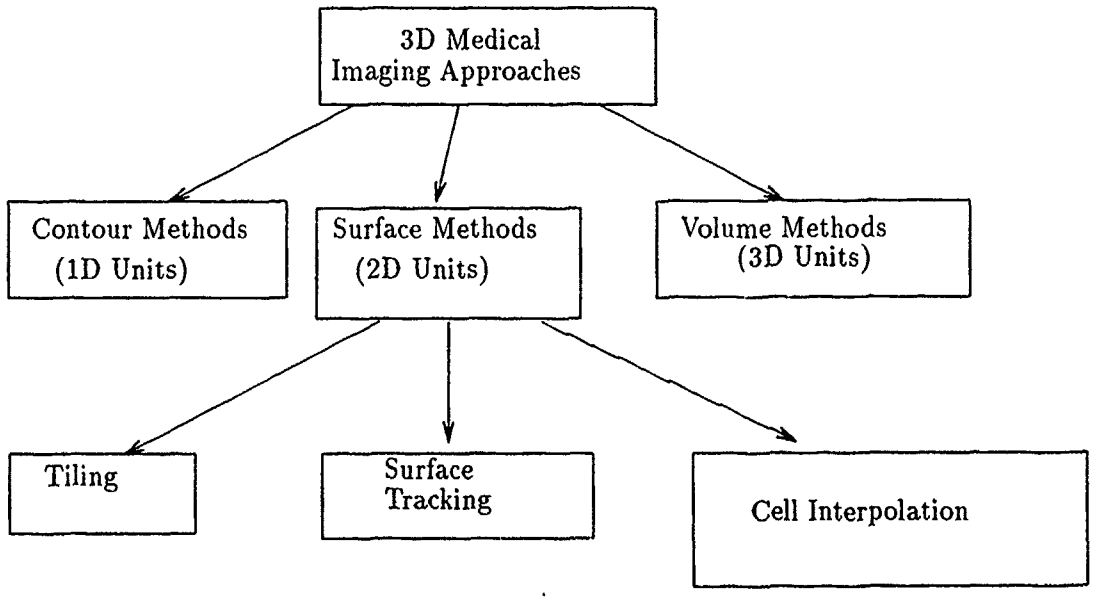


Figure 2.3. Classification of 3D Medical Imaging Methods

medical imaging topics that most directly relate to my work are discussed in this section. Rendering (projective transformations) is a major 3D imaging task, but is not discussed at length here because this research emphasizes estimation of scalar values in one form of "structure extraction" and the scene transformation operation of slice interpolation. (45) and (41) discuss the different 3D medical imaging rendering processes.

All 3D medical imaging techniques assume scene space consists of a 3D array of values (27). The 3D array is actually stacked slices of scanner generated data (see figure 2.1) with inter- and intra-slice thicknesses. This format will remain following any scene transformations.

Two processes that must be accomplished to perform structure extraction are segmentation and boundary detection. Segmentation partitions object space into objects of interest (which may or may not be surfaces) and the remaining volume. Boundary detection or surface tracking are techniques that locate surface boundaries (e.g., organs or bone) during image segmentation (40). These processes are referred to in the discussion of surface methods, but are mentioned here because they are done in both volume and surface methods.

The remainder of this section provides a very brief overview of the contour and volume methods and discusses the surface methods in more detail. For a complete discussion of these different approaches see (19) and (41).

2.3.1 Contour Methods The earliest approach to 3D visualization creates 1D contours (1D units). contours represent the border of the object of interest. Herman et al. (26) gives an excellent description of contours, along with a figure that clearly depicts a common 1D unit-based rendered surface. A contour corresponds to the surface border of the object on one particular 2D slice of the data. A contour is usually drawn as an approximated curve. When the contour lines for all the slices are rendered, the appearance of a 3D object surface is created. Contour lines are extracted manually, automatically, interactively, or by a combination of the three methods.

The obvious disadvantage of contours is the poor representation of the surface, because the only information depicted is in the plane of the slice and within the plane, only the surface boundary of the object. However, 1D unit-based images are generated and rotated rapidly. Due to the latter fact, they have found wide acceptance and use in the medical imaging community.

2.3.2 Volume Methods These techniques are volume oriented because they use 3D volume units as display primitives. Volume techniques differ from surface and contour methods by the amount of data that must be stored and processed during image computation. Additionally, volume methods typically preserve data continuity between surface boundaries, whereas surface and contour methods do not.

Surface rendering algorithms assume data consists of objects with thin surfaces in a volume of air (i.e., surfaces are easy to find, with little noise), whereas in reality most objects have fuzzy (thick) borders and there is much more than just thin air between surface boundaries. Volume methods preserve fuzzy borders and inter surface material by avoiding simple classification schemes that assign binary values to voxels indicating the voxel is in or out of the image to be rendered (33). The phenomenon that a voxel may contain more than one scalar value is termed a partial volume artifact. Volume methods allow percentages of different scalar values as well as color,attenuated light and /or transparency to be assigned to a single voxel

(16). Each voxel contributes to the final image based on these percentages, thus reducing the effect of partial volume artifacts. The final color of a pixel becomes the contribution of all voxel values lying along a ray's line of sight or along a projection path. Colors are weighted by transparencies and attenuation. Since every voxel contributes to the final image in this way, volume methods capture transitional areas between surface boundaries that might otherwise be missed by surface methods.

2.3.3 Surface Methods Surface methods attempt to reduce the volume of data to surface boundaries by depicting these boundaries by common graphics primitives such as polygons, patches, or points (2D units). Surface methods process a small number of slices at a time, hence they had dominated imaging algorithms for many years since computing power was not sufficient enough, until recently, to handle the entire volume of data at once. These techniques consist of tiling, surface tracking methods, and cell interpolation approaches.

Tiling

Tiling (tessellation) methods take as input contours created from any of the contour approaches. The next step usually filters the contour data by smoothing or re-sampling and then polyhedra such as triangles or quadrilaterals connect adjacent contours. Smoothing is done to better approximate the curve nature of contours. One heuristic-based method (42) uses B-splines to approximate a closer fitting contour, so contour smoothing is unnecessary. The following paragraph describes some of the tiling techniques developed in the past.

Tiling methods fall into two general classes, optimal based and heuristic based. The optimal based solutions (23) and (30) apply graph theoretic methods to derive what the authors consider the optimal triangulation between two adjacent planar contours. The major disadvantage of the optimal tiling methods is the long search time required to find the best triangulation. However, the process is entirely automatic. Since speed is an issue in medical imaging, many other methods were developed based on hueristics to achieve a faster tessellation, adding interactive assistance if needed for ambiguous cases (5),(42), and (24).

One major advantage of tiling is that it produces conventional graphics geometric primitives that can be rendered by applying standard reflection and shading techniques. In addition, as with contour methods, rapid viewpoint changes are possible and the data size in the final imaged data set can be quite small compared to the original volume of data.

Gross inaccuracies in an image can occur if contours are not well-formed with respect to each other. For example, this can happen when more than one contour is formed on a scan plane to represent a surface. The most well-known remedy is interactive editing, although it is time consuming and still error-prone. Even without interactive editing, automatic edge tracking to find the contours can be too slow for most applications. Speed of contour formation is proportional to the number of structures in the data set. Another major disadvantage is that tiling based on contours results in loss of essential information because contours do not contain enough gradient data to represent the actual surface.

Surface Tracking

Surface tracking methods generate a surface as a set of cuberille faces. Recall from chapter one that a cuberille is a dissection of 3D space into equal size cubes by three orthogonal sets of equally spaced parallel planes. This is a natural extension of the 2D space dissection forming quadrilles, or "square-shaped pixels" (4:34). The output primitives are planar polygons formed by connected cube faces approximating a surface of interest.

The first step usually accomplished in surface tracking methods is to modify the volume to form cubes. This is done by interpolating values in the components needed (19:327). For example, Artzy et al. (1:19), Artzy (2:6), Udupa (44:220-221) linearly interpolated in one dimension so their input data would have an interslice distance equal to the resolution of the original 2D slices. This works as long as the resolution of the original 2D slices is square. If not, interpolation in two or three dimensions might be necessary.

Prior to rendering the cube faces, two tasks must be accomplished. First, the voxels must be segmented into those being in the object(s) of interest (1-voxels) or out (0-voxels). This binary classification forms a 3D binary volume (45:49). Next, the surface boundary(s) located between 1- and 0-voxels must be located and display elements connected. The term surface tracking is derived from this process of locating the surface boundary from a binary volume.

Artzy et al. (1) developed a surface tracking algorithm that reduces the challenge of finding connected voxels representing the surface boundary to a graph traversal challenge. Voxels are first segmented using binary classification. Next, boundary detection is accomplished. Nodes of a directed graph, G, then correspond to voxel faces separating the object under interest from all else in the scene. The authors prove that connected subgraphs of G correspond directly to surfaces of connected components of the object. To find the surface boundary, a subgraph of a digraph is traversed.

Cell Interpolation Approaches

Cell interpolation methods generate polygonal elements by analyzing computational cells. Recall from chapter one that a computational cell is a parallelepiped such that four cell vertices are voxels in one slice and the other four are voxels in an adjacent slice - see figure 2.4. The major difference between these methods and the cuberille methods is that the cell interpolation methods analyze how the data varies between voxels to determine where the surface lies versus assuming only constant or linear variation.

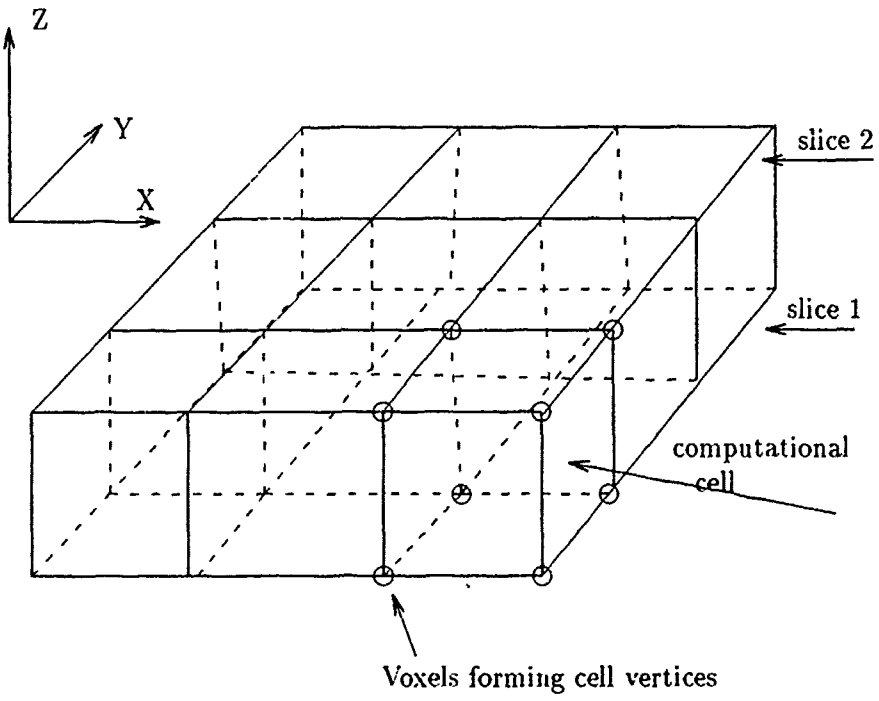


Figure 2.4. Computational Cell

There are currently four types of cell interpolation approaches. These algorithms are the marching cubes method developed by Lorenson and Cline (34). the soft objects method by Wyvill and McPheeters (52), the "gradient-consistency heuristics" by Wilhelms and Gelder (50). and the cell subdivision techniques also described by Wilhelms and Gelder (50).

Each method follows a two step process. First, the volume of data is segmented by classifying each voxel as either 1 or 0 (or in the case of the soft objects method as

hot or cold). A voxel is 1 if its scalar value is greater than an iso-value (threshold), else it is assigned a 0. The second step is to determine if a cell has both 1 and 0 vertices and if so generate polygons within the cell to approximate the portion of the iso-surface that passes through the cell. An iso-surface is formed by connecting all the polygonal elements to form a 3D mesh, such that the surface intersects approximately the same (iso) scalar value or range throughout the sample data. Polygon vertices (not to be confused with cell vertices) are determined by linearly interpolating 3D coordinates between the 1- and 0-voxels of a cell. The coordinates are interpolated to the iso-value. The four methods differ by how the polygons are formed within the cells.

The term ambiguous cell must be defined before discussing the four cell interpolation methods. A cell is termed ambiguous if more than one topology can be chosen for it. A topology is the polygon formation within a cell. Durst (18) noticed holes can result from the marching cubes algorithm described by Lorenson and Cline. Holes can be caused by improperly forming polygons between two ambiguous cell faces.

The term ambiguous cell was defined by Wilhelms and Gelder (50). An ambiguous cell face is defined as "a cell face that contains a diagonally opposite pair of positive vertices [1-voxels] and a diagonally opposite pair of negative vertices [0-voxels]" (see figure 2.5, obtained from (50)). By looking at figures 2.6, 2.7, and 2.8, the reader can see that the ambiguous cases are 3, 6, 9, 12, 13, and 14.

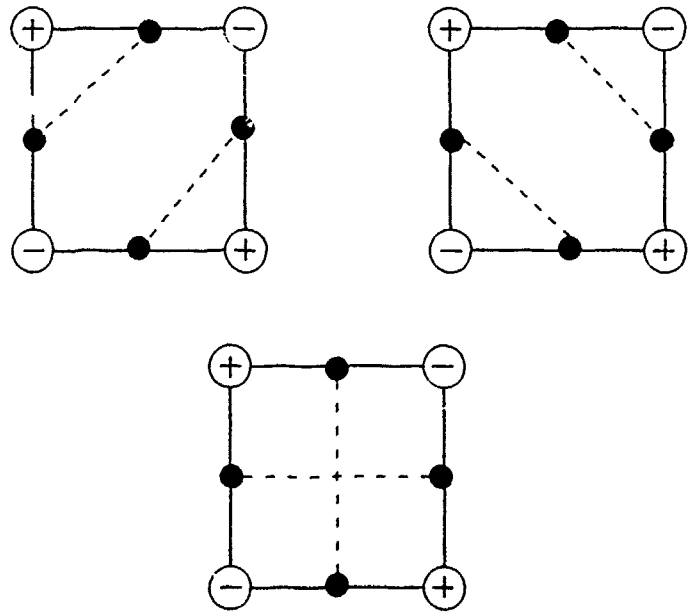


Figure 2.5. Ambiguous face - intersection point connection choices

Marching Cubes

The marching cubes algorithm was developed to alleviate the need of using cubic voxels, i.e., 3D data " with reduced resolution in one dimension" (7:345). Binary classification of cube vertices creates a total of $2^8 = 256$ possible cell vertex classifications. By analyzing the geometry of the different cases, the total number of unique cases can be reduced to only 15 (see figures 2.6, 2.7, and 2.8). The other 241 cases are reduced to the 15 by symmetry and appropriate rotations. Wilhelms and Gelder (50) call this approach the major case table lookup method because a 256 element table must be pre-set to indicate the transformation of cases to the unique 15. The signs at the cell's eight vertices are then used as an index into the major case table. Once the appropriate classified cube case is determined, another pre-set table entry indicates the triangle formation. The triangle formation is somewhat arbitrary since intersection points (points approximating where the iso-surface intersects a cell edge) can be connected in many different ways for most of the 15 cases. Only certain connections make sense with most of the cases. However, there are six cases that can

cause serious inaccuracies if improperly connected (which is discussed in the next section).

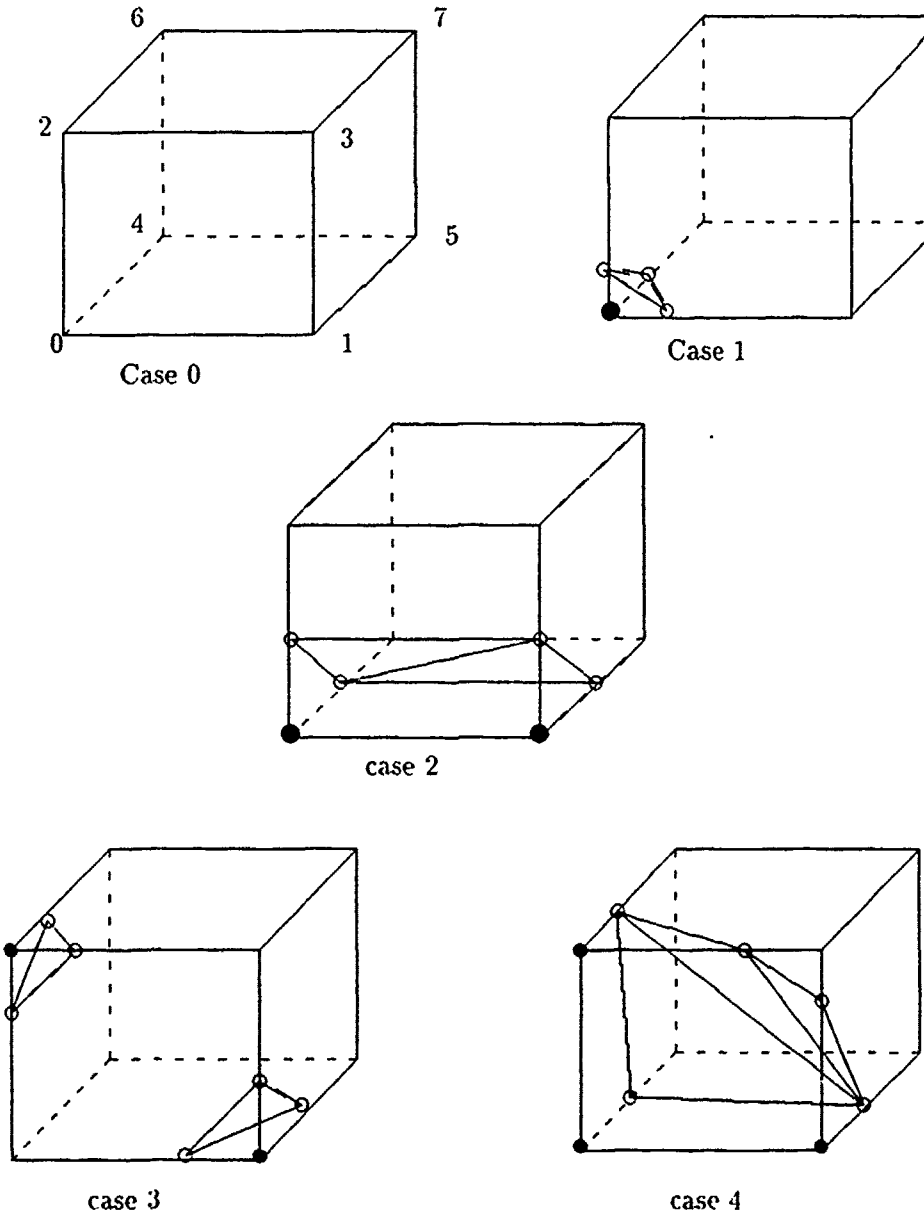


Figure 2.6. First 5 cases of 15 cube vertex classifications and triangulations

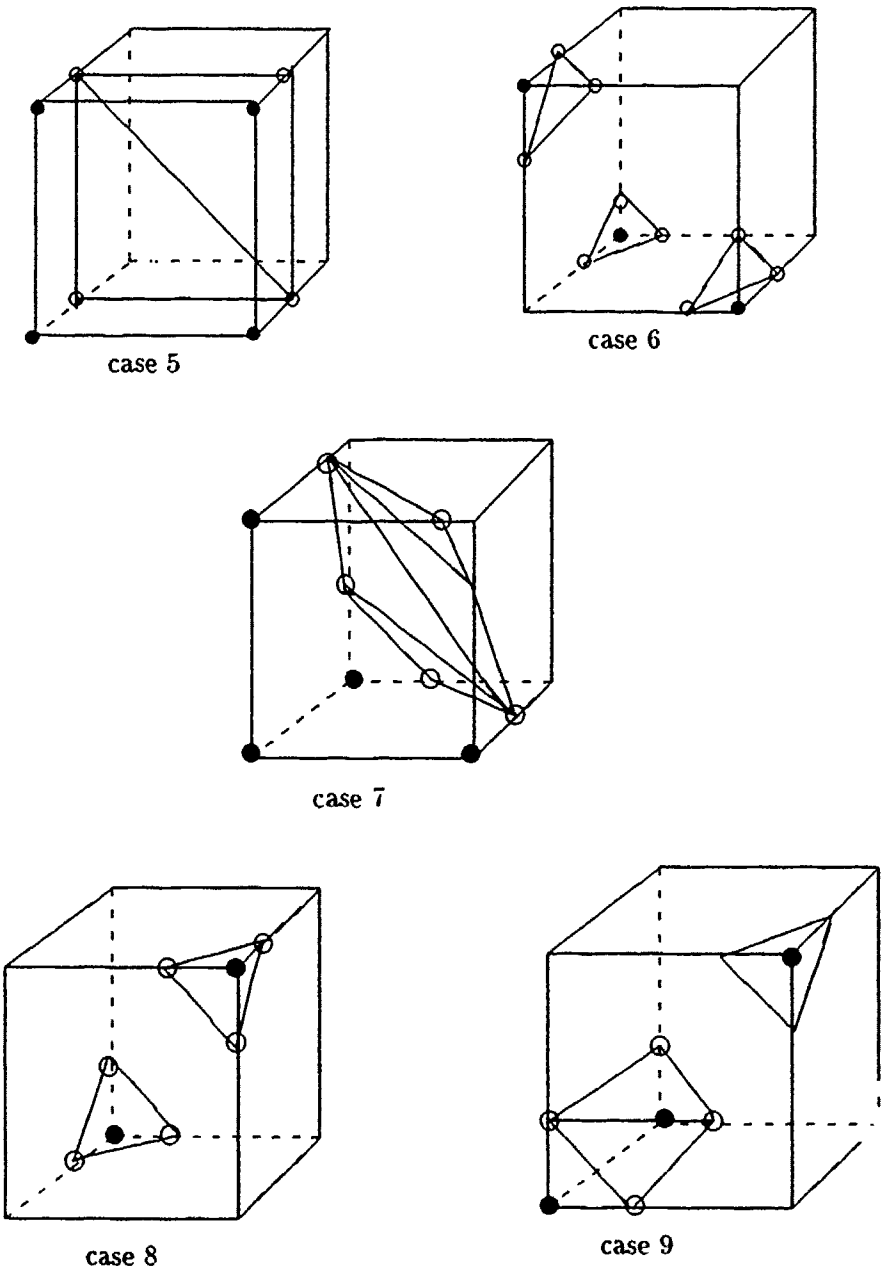
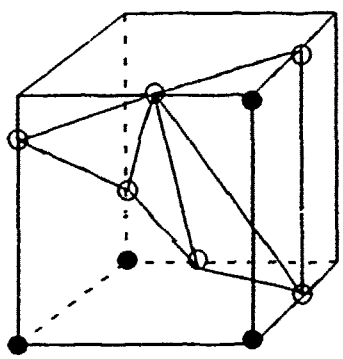
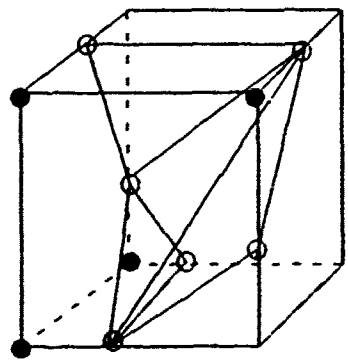


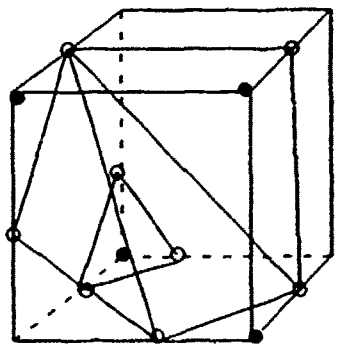
Figure 2.7. 5 more cases of 15 cube vertex classifications and triangulations



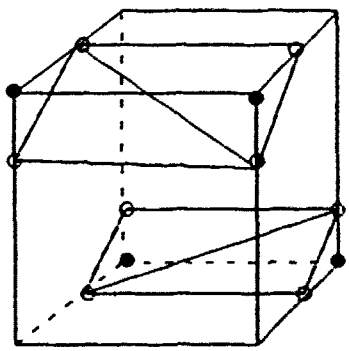
case 10



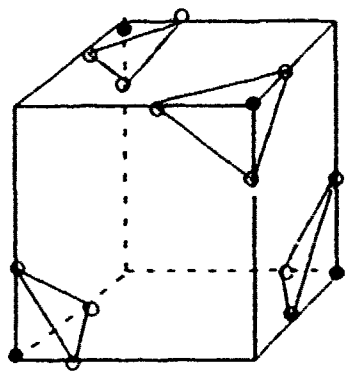
case 11



case 12



case 13



case 14

Figure 2.8. Final 5 cases of 15 cube vertex classifications and triangulations

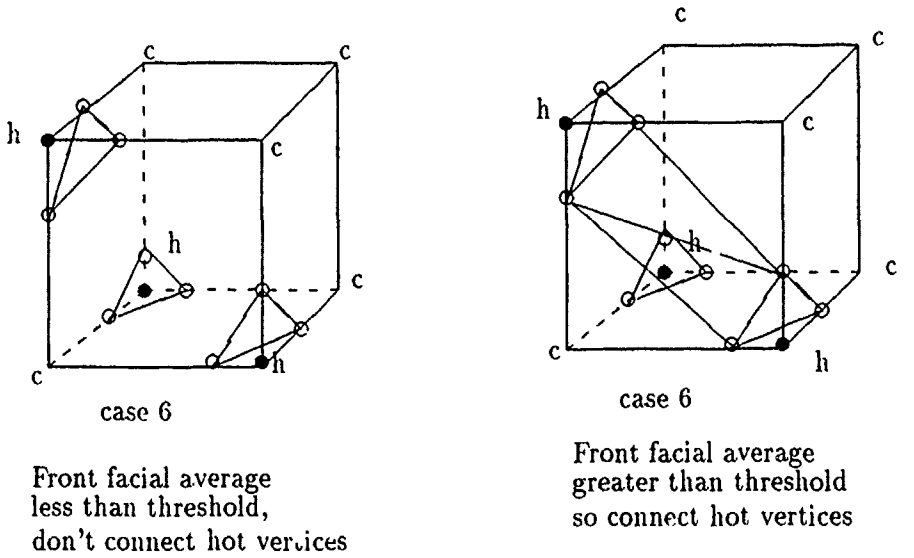


Figure 2.9. Facial averaging example

Soft Objects

Wyvill and McPheeter's soft objects method is almost identical to the marching cubes algorithm. The actual 3D scalar field is fabricated rather than obtained from scanners, but the data representation is still a 3D regular data set. Key points are specified to reduce the data set. Additional points are estimated as needed by a cubic function that uses a radius of influence to determine the key points needed in the estimation function. The most significant difference between this method and marching cubes is that this method uses a dynamic, simple technique to polygonize an ambiguous cell.

The soft objects method forms polygon vertices on an ambiguous cell face by analyzing the four cell face vertices. The method assumes the value at the center of the face is approximated by averaging the four vertex values. Then, if the averaged value is greater than the iso-value, the positive vertices are connected. For example, see figure 2.9. In this way, two cells sharing ambiguous faces are always consistently polygonized – thus no holes result.

This facial averaging method provides only a rough estimate of center face values. A different form of estimation may be required to obtain a more accurate center value estimate. Even if the estimate is not very accurate, Wilhelms and Gelder claim the facial averaging method guarantees continuity, i.e., no holes appear in the image. This assertion is based on their facial plane principle :

> If the method of disambiguation for ambiguous faces employs only values in the plane of the face, and is invariant under rotations and mirror images, then the isosurface as defined by topological polygons will be continuous.

They refer to the (possibly nonplanar) polygon generated within a cell as a topological polygon because it specifies "the topology of the isosurface within the cell." This assertion makes intuitive sense because the same facial average is calculated for shared cell faces. This results in shared vertices between cells for that face.

Gradient Consistency Hueristics

The gradient consistency hueristics were developed by Wilhelms and Gelder to compute a better estimate of the center face value, assuming that simple averaging does not estimate the center value accurately enough (50). The authors use these methods to handle ambiguous cases in the same manner as facial averaging. The authors use the gradients at the four face corners to obtain the center estimate. Gradients indicate how rapidly the iso-surface is changing at a particular point. The magnitude of the gradient is used as an approximation to the normal in the shading calculations, therefore it is already computed at each voxel. The two gradient consistency hueristics developed by the authors are the "center-pointing gradient" and "quadratic fit."

The center-pointing gradient is the gradient component of a cell vertex directed towards the cell face center. This gradient is obtained as the dot product of the vertex gradient with a unit vector in the direction from the vertex towards the center of the face. Two univariate quadratic functions are determined by the least squares error method to estimate the center scalar value along both diagonals of the face. These two estimates are then averaged to yield a final estimate. This estimate is then used to determine the cell topology, as in the soft objects facial averaging method.

Two assumptions are made in this method. First, the center-pointing gradients are assumed to approximate the derivatives at each corner. Second, the quadratic functions are assumed to exactly fit the two endpoints. The gradient assumption is typically used for determining the normals for shading as well. The second assumption is made in many estimation methods to insure the estimation process returns the correct value at known points.

This method only analyzes the face corner sample values, similar to the soft objects method. It should choose the topology better than the soft objects method only if the underlying scalar field function is quadratic along both face diagonals.

The quadratic fit method is similar to the center-pointing gradient method, except here the center face value is estimated by a single bi-variate quadratic function. The least squares error fit uses all four face corner sample values to estimate parameters of the function. This method chooses a more correct topology than the center-pointing gradient if the underlying scalar field function is quadratic across the entire face.

The gradient consistency heuristics make minor improvements to the soft objects method of resolving ambiguous cells. A disadvantage of the heuristics methods is that they assume the scalar field function is locally quadratic. If wrong, this assumption can generate inaccurate topologies within cells. The only way to know if the topology is inaccurate is to know the local variation of the scalar value function. However, obtaining this knowledge might be too computationally costly to be of any benefit.

Cell Subdivision Techniques

Cell subdivision techniques are implemented for two primary reasons. The first is to increase image fidelity. The second is to resolve ambiguity of cells. Image fidelity can be increased because the data resolution is increased. Cell subdivision is the same method employed by Cline and Lorensen's (8) dividing cubes algorithm. In dividing cubes, cubes (cells) are divided until the resolution of point primitives is reached. In cell subdivision, instead of subdividing cells to create point primitives, cells are subdivided to create sub-cells. These sub-cells are then treated as the previous original cells were, i.e., the iso-surface is represented within them by triangles. These sub-cells are much smaller than the original ones, hence data resolution has increased. Because of this increased data resolution, the surface can be approximated closer to the actual surface by the smaller triangles. However, the quality of the final image depends on how the scalar values are estimated at the sub-cell vertices. Therefore, the significant challenge with any subdivision technique is how to estimate values at the sub-cell vertices.

According to Wilhelms and Gelder, another way to resolve ambiguity in cell cases is to create sub-cells within original cells and polygonize the sub-cells as in the soft objects method. New cell vertex scalar values are estimated by a re-sampling function. In Wilhelms and Gelder's implementation, they subdivide each computational cell and apply an estimation function to determine new scalar values at the points derived from the subdivision. The two estimation functions they use are trilinear and tricubic. The major case table lookup method (marching cubes) is used to process the sub-cells, and ambiguous cases are handled as in the facial averaging technique. Tricubic is more computationally intensive than trilinear, but it produces more accurate images for their artificially created volumes than any of the other cell interpolation methods. Image quality improves because the tricubic method considers sample data in a large neighborhood (the surrounding 64 voxel values) and does not assume linear variation. On the other hand, trilinear only analyzes the eight cell vertex values and assumes a linear variation along each of the three axes.

Note that all nonempty[1] cells must be subdivided to ensure continuity between faces. If not, one large undivided nonempty cell face may share a face with several sub-cell faces.

Besides just resolving ambiguous cases, cell subdivision is also a good method to use if a smoother looking image is desired. However, depending on the subdivision size, the resulting number of polygons may be very large. A typical set of from 50 to 100 brain MRI or CT slices can result in over 500,000 polygons for certain iso-values. Subdividing by a factor of just two could increase this to over two million polygons. In this case it might be more beneficial to use Cline and Lorensen's Dividing Cubes algorithm (8).

[1]Nonempty means that a classification of the vertices derives unique cell case 0 in figure 2.6. That is, a nonempty cell is one such that the surface is determined to not pass through it

Surface Methods Conclusion

Both the cuberille-based methods and the cell interpolation methods produce images that appear to represent the surface of interest. However, the most important issue is not appearance but rather the accuracy of the methods. An argument could be made that the pre-processing step of creating cuberilles produces more accurate results because of the increased resolution. However, new slices are normally created by simple linear interpolation, whereas a higher order interpolation might be more accurate. Also, once the cuberilles are formed, constant variation within cells is assumed during surface formation.

Cell interpolation techniques using subdivision and estimation functions also estimate new scalar values within the original computational cells and can easily go beyond input pixel resolution by increasing the subdivision factor. In addition, according to Upson and Keeler (46), the variability of the scalar values within cells "..is a more accurate representation of the real world and allows for smooth, continuous representations of even small datasets." Cell interpolation methods assume variability within cells to create new sub-cells. They also assume scalar value variation within the new sub-cells (although this latter variation is always linear – recall the process of finding surface intersection points along cell edges).

An advantage shared by the cuberille-based and the cell interpolation methods is that they create a display list of polygons. This display list can be saved for fast re-display at different viewpoints using traditional lighting and shading models.

A major disadvantage to the marching cubes cell interpolation algorithm is that holes can appear in the rendered surface (18). Yet, with disambiguation methods employed, the ambiguous cases are corrected to a certain degree. Cell subdivision methods can also improve image quality as well as the accuracy of surface representation. Both image quality and accuracy of surface representation depend directly on the accuracy of the estimation function.

2.4 Chapter Summary

3D medical imaging topics relating to this thesis were presented. Specifically, this chapter first discussed the data representations used in 3D medical imaging. Regular data grids are assumed by all 3D medical imaging methods because the output from medical imaging modalities is regular. The two primary data models

include voxels and cuberilles. Next, the common 3D medical imaging transformations were summarized. The remainder of the chapter described the 3D medical imaging methods - contour, volume and surface. Surface methods were covered in great detail because surface methods were implemented to accomplish the purpose of this thesis. The first primary surface method discussed is surface tracking. Since these techniques use the cuberille data model, surface tracking methods usually estimate new slices between existing slices to form cuberilles. Although a surface tracking method was not implemented in this research, the task of slice interpolation was accomplished. Finally, cell interpolation surface methods were discussed. These include the marching cubes algorithm, the soft objects method, the gradient consistency heuristics, and the cell subdivision techniques.

The focus of the next chapter is to explain how an estimation function can be derived based on the statistics of the underlying scalar field data.

III. Kriging Theory

Kriging is a geostatistical estimation method named after a South African mining engineer, D. G. Krige. The theory was developed for the purpose of estimating ore reserves in mining. However, kriging can be used to estimate samples in any situation where estimates depend on nearby known samples based on spatial or temporal position. Kriging is significant because it is the optimal linear estimator.

This chapter presents first the background and purpose of kriging, followed by the derivation of the ordinary point kriging equations. After these equations are derived, different kriging categories and types are summarized. Lastly, an overview of structural analysis and isotropy is presented. The order of topics was chosen to present the most common form of kriging first, and then building on this foundation, to introduce other kriging methods.

3.1 Background

Krige developed the basic theory, but a French engineer named Georges Matheron and his colleagues developed the rigorous mathematical theory of kriging (35:602) and (9:625). Prior to kriging, estimation methods used in geo-statistics made several simplifying and usually invalid assumptions. The most erroneous assumption was that variances between data samples is constant. This assumption made the other estimation methods very error-prone. Krige pointed out that to get a more accurate estimate, variances between the prospective blocks and the core samples must be taken into account. Kriging is primarily used in situations where there is expected to be some dependence between data measurements at different locations. Its use in discovering deposits in various mining pursuits is well known and documented, see (13:70-71).

Kriging is a process that derives a geo-statistic. Geo-statistics differ from classical statistics in the variables used. Recall that a statistic is a function of random variables. A geo-statistic is a function of regionalized variables. Random variables model chaotic processes. Regionalized variables model spatially dependent natural phenomena. According to Matheron (35), regionalized variables are characterized by three qualities. The first is localization. Regionalized variables are localized within a support. A support is the volume of a sample, consisting of geometric

size, shape, and orientation. In the geo-sciences, an example support is a drill core. The second quality of regionalized variables is that they may exhibit continuity within the region of a support. Statistical continuity means the sample values do not deviate significantly from each other. Thus, they are not random in nature, but show some kind of order. The third characteristic of regionalized variables is anisotropies, discussed later in this chapter.

Kriging is a modified form of a multiple linear regression model with parameters estimated by a technique similar to the method of least squares (37) and (15). Kriging uses weighting functions based on distance to compute the desired data value. The method operates on the assumption that data points closer to the target should be weighted heavier in the estimation calculation than those further from the new point. For example, in figure 3.1 points 1,2, and 7 would be expected to have more influence on the estimate than points 5 and 6. This weighting strategy has

Figure 3.1. Kriging Example

proven to be very accurate in the field of geo-statistics. Davis sums up the goal of kriging in the following sentences :

> There are an infinity of other possible combinations of weights that could be chosen, each of which will give a different estimate and a different estimation error. There is, however, only one combination that will give a *minimum* estimation error. It is this unique combination of weights that kriging attempts to find.

The goal of kriging is to estimate an unknown value at a particular place within a known neighborhood of points using a linear combination of computed weights and known sample values. Optimal weights are determined by solving a system of linear equations produced by two conditions placed on the kriging linear combination. These constraints make the value derived from kriging the best linear unbiased estimator (B.L.U.E.) (13). The constraints are:

- unbiasedness by setting the expected estimation error to zero i.e., removing any "systematic error"
- minimum estimation error variance.

These conditions insure optimality and insure that "no other linear combination of the observations can yield estimates that have a smaller scatter around their true values" (14:385).

Why use a linear estimator versus a non-linear? According to Delfiner and Delhomme, "The major difficulty with non-linear estimators is that they involve parameters or characteristics that cannot be inferred from the data" (15). Even if the sampling function is locally non-linear, this is taken into account by the drift in universal kriging, which is discussed later.

The basic purpose of kriging for this research is the same as that for the estimation functions explored by Wilhelms and Gelder (50) - given a new point within a neighborhood of known sample points and associated values, estimate a value at the given point using a combination of known sample point values. The kriging equation estimates a value as a distance-weighted linear sum of known sample points. The term linear refers to just the linear sum, it does not indicate that the data varies linearly. Kriging can model all forms of higher order data trends such as quadratic and cubic. This capability is discussed in later sections. The next few sections explore how kriging calculates data weights and show that the conditions assumed about the data make kriging the optimal interpolator for data sets derived from natural phenomena.

3.2 *Kriging and Least Squares*

Kriging primarily differs from least squares in the type of variables used in the linear equation. The variables used in least squares are assumed to be independent random variables, whereas those in kriging are regionalized variables. Recall from a previous definition that regionalized variables model natural phenomena. These types of variables assume the data is localized and exhibits continuity, whereas phenomena modeled by random variables exhibit chaotic behavior. No simple methods like least squares can be applied to such variables because no tractable deterministic functions can be found that describe the complex variations in regionalized variables (14). The next section discusses how the dependence of regionalized variables is captured in the kriging linear regression model in the simplest kriging case.

3.3 *Derivation of the Ordinary, Point Kriging Equations*

My discussion of kriging begins with ordinary point kriging because the other forms are modifications of this method (ordinary and point are defined in the next section). The discussion begins by stating the goal of kriging, followed by an explanation of the constraints on the kriging equation that produce a system of equations. The system of equations are solved to yield the weights in the kriging equation. The equations presented below were obtained from a number of different sources, primarily (12), (14), (17), (9), (15), (13), (48), (11), (31), and (6).

In the following equation the goal is to estimate $\hat{Z}$, the unknown value at the known position p in the neighborhood of known points p_i and known values $Z_i(p_i)$.

$$\hat{Z}(p) = \sum_{i=1}^{n} w_i Z_i(p_i) \qquad [1]$$

The Z_i's are the regionalized variables with the parameter being an n-dimensional point and the w_i's the weights. The weights are chosen to satisfy the following two conditions that make $\hat{Z}$ the B.L.U.E. :

- $E(\hat{Z} - Z) = 0$ [2]
- $E(\hat{Z} - Z)^2$ minimum [3]

where $\hat{Z}$ is the value being estimated at p and Z is the actual value at point p. The estimation error, $\hat{Z} - Z$, is a measure of the dissimilarity between the two variables $\hat{Z}$ and Z. $E(\hat{Z} - Z)^2$ is the mean square error and E is the expected value or mean.

Using the following equality from the definition of the variance, V, (37:89),

$E(\hat{Z} - Z)^2 = V(\hat{Z} - Z) + [E(\hat{Z} - Z)]^2$

[3] can be re-written as

$\sigma_e^2 = V(\hat{Z} - Z)$ minimum [4]

where σ_e^2 is the estimation or error variance. This is important because it points out that even though condition [3] states minimum mean square error, it is equivalent to minimum estimation error variance.

Next, the system of kriging equations are derived from conditions [2] and [4]. This system is similar to the set of simultaneous linear equations (normal equations) produced in linear regression that are formed by setting the partial derivatives of the unknown parameters to zero (21). Before the equations can be derived, the above two conditions are expanded and changed into more quantifiable constraints.

Modifying condition [2] above is straightforward :

$$E(\hat{Z} - Z) = 0$$

$$E(\hat{Z}) - E(Z) = 0$$

Then recalling [1], substitute into the above and get as an additional constraint

$$E[\sum_i w_i Z(p_i)] - E(Z) = 0$$

$$\sum_i w_i m(p_i) - m(p) = 0$$

$$\sum_i w_i = 1 \qquad [5]$$

where $m()$ is the mean or first moment. Unbiasedness in the estimate is assured by insuring that the weights sum to 1. (13:238)

The estimation error variance is ((6), (13), and (9))

$$\sigma_e^2 = 2\sum_i w_i K(p_i, p) - K(p, p) - \sum_i \sum_j w_i w_j K(p_i, p_j) \quad [6]$$

where K(m,n) is the covariance between point m and point n and p is the n-dimensional point where the estimate is computed. Journel derives similar estimation error variances (29), but this equation is considered the general unbiased linear estimator derivable by expanding the variance of the linear combination of regionalized variables in equation [4].

The covariance between the variables is modelled by a function called the semivariogram. In most cases the semivariogram is unknown and must be determined

by a process called structural analysis. For the present, assume the semivarogram is known and is represented as $\gamma(m,n) = \gamma_{mn}$. The semivariogram represents the average difference squared between the values at points m and n. The main parameter used in semivariograms is the distance between points m and n. The semivariogram models the dependency of data values based on how far apart they are from each other. The spatial distribution of regionalized variables is accounted for in the semivariogram. The semivariogram gives the correlation between sample values a geometric meaning rather than a probabilistic meaning. Structural analysis and the semivariogram are discussed in more detail in a later section. As a result of substituting the semivariogram in place of the covariance, [6] becomes :

$$\sigma_e^2 = 2\sum_i w_i\gamma_{ip} - \gamma_{pp} - \sum_i\sum_j w_i w_j \gamma_{ij} \qquad [7]$$

Now that condition [4] is more quantifiable in terms of [7], it must be minimized to satisfy the minimum estimation variance constraint. Since there is a constraint [5], n unknowns and n + 1 equations would result from minimizing this system. Therefore a Lagrangian Multiplier is added to equalize the system. Minimization is done by taking the partial derivatives with respect to the weights and the Lagrangian Multiplier η and setting the resulting equations equal to zero. This yields the following complete ordinary point kriging system:

$$\sum_{j=1}^{n} w_j\gamma_{ij} + \eta = \quad (i = 1,\ldots,n) \qquad [8]$$

$$\sum_{j=1}^{n} w_j = 1$$

The Methodology chapter will show these equations in expanded matrix form.

The system presented above is ordinary point kriging. Due to the constraints [2] and [4], this system will determine optimal weights to substitute back into equation [1]. $\hat{Z}$ is the B.L.U.E. because of the optimal weights, i.e., no better linear estimate can be derived. This system is only one of several types of kriging possible. Other types and their differences from this system are discussed next.

3.4 Kriging Categories and Types

The kriging literature describes two broad categories of kriging and three types of kriging. The categories differ based on estimation region. The types differ based

on assumptions about the sample data. Either of the two categories can be used in any kriging type.

3.4.1 Kriging Categories The two categories of kriging are point and block. Point kriging was discussed in the preceding section. It is employ. when the goal is to estimate a value at one particular point.

Block kriging estimates a value for a region instead of at a single point. There are two block kriging methods. The first uses point kriging repetitively to estimate several values within the block and then averages the results to get one value. The second method derives a new set of equations using a modified covariance function of the K() terms in equation [6]. The second method involves computing a double integral that evaluates the area of the block in question. The problem with the second approach is finding an explicit analytic form for the integral; hence, the first method is most often used for block kriging (11)[1].

3.4.2 Kriging Types There are three primary types of kriging discussed in the literature - simple, ordinary and universal. These techniques differ in their assumptions about the behavior of the expected values or means of the regionalized variables $E(Z_i)$ (15), see figure 3.2. Each of the three methods can be developed so the final kriging system estimates points or blocks.

[1]Some authors combine point and block kriging into one form in which the area integral reduces to a point in the case of point kriging (9:626)

Kriging Type	Means of Regionalized Variables are Known	Means of Regionalized Variables are Constant
Simple	Yes	Not applicable
Ordinary	No	Yes
Universal	No	No

Figure 3.2. Table of Kriging Types (All three can estimate points or blocks)

Simple Kriging

This type of kriging, as the name suggests, is the simplest form of kriging – even simpler than the above ordinary kriging system. However, this method is seldom employed. The method is termed "simple" because sample means at known locations are assumed to be *known* prior to kriging. The means are stated as

$$E(Z_i) = m_i, i = 1, \ldots, n \text{ and } E(\hat{Z}) = m_z$$

where m_i is the mean of the i'th regionalized variable and m_z is the mean of the estimator regionalized variable.

This assumption modifies the kriging equations derived above because the unbias constraint changes. The ordinary equations derived above left out one variable important to the simple kriging equations. This variable is called a shift parameter, λ (29). In ordinary kriging, $\lambda = 0$. The shift parameter modifies the estimation as

$$\hat{Z} = \lambda + \sum_{i=1}^{n} w_i Z_i(p_i)$$

Now the unbias condition becomes :

$$E(\hat{Z} - Z) = E(\hat{Z}) - E(Z) = E(\sum w_i Z_i(p_i) + \lambda) - m_z$$

Journel (29) shows that this unbias constraint changes the estimation equation to

$$\hat{Z} = m_z + \sum_{i=1}^{n} w_i (Z_i - m_i)$$

This equation indicates that the unbiased estimator $\hat{Z}$ is determined as a linear combination of the residuals, $(Z_i - m_i)$.

This assumption also simplifies the minimum error variance condition [4]. In essence, this form of kriging reduces to classic linear regression (29) and (12). Simple kriging is seldom used because the means of the regionalized variables are usually unknown.

Ordinary Kriging

Unlike simple kriging, ordinary kriging assumes the mean of each regionalized variable is *unknown*. Also unlike simple kriging, ordinary kriging assumes each mean is the same. A constant mean is more commonly referred to as a stationary mean (9:626). The ordinary system of equations were derived above to estimate points.

Universal Kriging or Kriging with a Trend

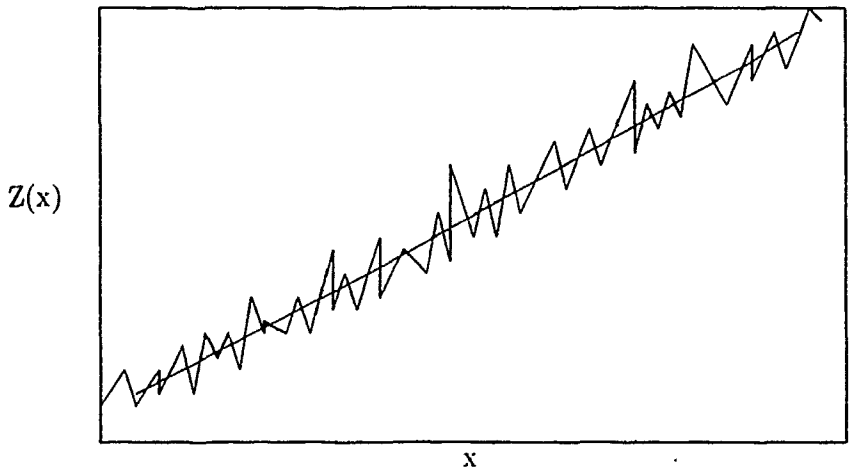
Universal kriging also assumes unknown regionalized variable means; however, it differs from ordinary kriging because it assumes the unknown sample means are possibly different, or non-stationary. Sample means are also referred to as first-order moments. Stationarity assumptions for the second-order moments can also impact the type of kriging used (see (9) and (11)), but will not be considered in this effort. The term non-stationary also refers to drift or trend, indicating the means of the regionalized variables differ from sample point to sample point within a surrounding neighborhood of other sample points. There are two primary methods of kriging in the presence of local drift - universal kriging and the method of intrinsic random functions. According to Cressie (9), universal kriging, simply extended from ordinary kriging, is most often used because the method of intrinsic random functions is impractical to apply.

When using universal kriging, the first process becomes estimation of the regionalized variable means at the sample points using a local neighborhood of known sample values to determine if the means are constant from sample point to sample point. If data has global (regional) drift, there appears to be a definite pattern or drift in the sample values over a larger area, usually much larger than the neighborhood size used in the kriging system. If drift is local, it occurs within the neighborhood. Figure 3.3, adapted from (14) and (13), shows regional drift as a line placed through the data points.

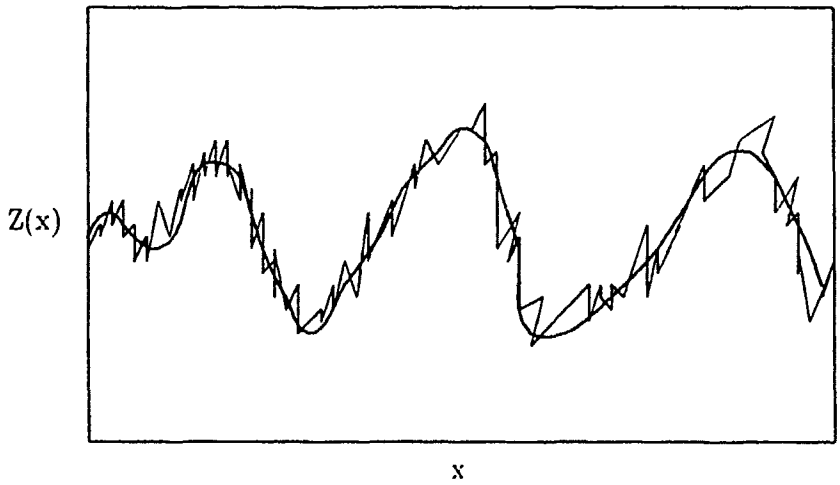
If global drift exists, the regionalized variables are now viewed as being composed of two parts - the *drift* and the *residual*. The drift of a regionalized variable is its expected value (mean) at a point p_i within a certain neighborhood. This experimental or computed mean is called drift if it varies from point to point. The residual is calculated by subracting the drift from the actual measurement. For example, assume an experimental drift is calculated for every known point[2], call it $m(Z(p_i))$. Then the residuals are calculated as

$$Y(p_i) = Z(p_i) - m(Z(p_i)).$$

[2]David shows that the experimental drift can be calculated by estimating the drift coefficients as a linear combination of the available data. This is another multiple linear regression that typically assumes simple unbiased estimation derived from a least-square method (13) The drift equation is shown in the next paragraph.



Global Trend and Local Stationary Mean



Local Trend and Global Stationary Mean

Figure 3.3. Global and Local Drift

There are three primary steps in estimating points or blocks in the presence of a global drift. First, the drifts are estimated at each point and residuals are computed. Then, the residuals are used as stationary regionalized variables in a simple, ordinary or universal kriging system (depending on the local means). Lastly, the estimates derived by kriging the residuals are added back to the drifts to get the final estimate.

If local drift exists, then universal kriging is used (13:267). The ordinary point kriging equations developed in the last section must be modified in the presence of local drifts to yield the system of universal kriging equations. These modifications are described below.

David (13) expresses the drift at point p as :

$$m(p) = E(Z(p)) = \sum_{l=0}^{k} d_l f^l(p) \qquad [9]$$

Equation [9] is normally represented as a finite order polynomial in universal kriging. The $f^l(p)$'s are $k+1$ known functions, usually monomials, and the d_l's are the $k+1$ unknown drift coefficients.

The kriging system of equations change in the presence of drift because the unbias constraint changes. In ordinary kriging the unbias constraint forces the weights to sum to 1 because the mean $m(p)$ is constant, producing equation [5]. Now that $m(p)$ is no longer constant but takes into account drift from equation [9], the unbias constraint results in

$$E(\hat{Z} - Z) = 0$$

$$E(\sum_i Z(p_i)) - \sum_k d_l f^l(p) = 0$$

$$\sum_i w_i f^l(p_i) = f^l(p) \qquad (l = 0, 1, \ldots, k) \qquad [10]$$

Notice the drift coefficients d_l have dropped out of the constraint. Thus the universal system is independent of the drift coefficients, but still insures unbiasedness. Since this condition insures unbiasedness regardless of the unknown drift coefficients d_l, the term universal is used to denote the system of equations that result.

This constraint (equation [10]) adds $k+1$ more equations to the minimum variance condition, thus $k+1$ additional Lagrange Multipliers (η_l) are needed (9).(15).and (17). After the partial derivatives of the equations are taken with respect to the n weights and the $k+1$ Lagrangian Multipliers and set to zero. the resulting universal kriging system is

$$\sum_{j=1}^{n} w_j \gamma_{ij} + \sum_{l=0}^{k} \eta_l f^l(p_i) = \gamma_{ip} \qquad (i = 1, ..., n) \qquad \text{minimum estimation variance [11]}$$

$$\sum_{i=1}^{n} w_i f^l(p_i) = f^l(p) \qquad (l = 0, 1, ..., k) \text{ non-bias [12]}^3$$

There are several unknowns in this system of equations that must be estimated. The unknowns in the universal system are the order k of the polynomial $f^l(p)$, the drift coefficients d_l, and the size of the neighborhood used to determine the drift. These unknowns are determined during structural analysis. The drift coefficients d_l can be found along with the weights in the kriging system (14:394) or by a least-squares method (13:272). The order k is usually 1 or 2. If the means of the regionalized variables are the same, $k = 0$ and equation [10] reduces to equation [5], which is ordinary kriging. If the order is 1 this means the 0th order term or the constant will be included as well as the first order terms. In two dimensions this is the x and y terms. If k is 2, the 0th, 1st, and 2nd order terms are included in the drift. The first order (k=1) polynomial associated with linear drift in the neighborhood is : $m(p) = d_0 + d_1 X_{1i} + d_1 X_{2i}$
and the second order polynomial associated with quadratic drift is :

$$m(p) = d_0 + d_1 X_{1i} + d_2 X_{2i} + d_3 X_{1i}^2 + d_4 X_{1i} X_{2i} + d_5 X_{2i}^2$$

where X_{1i} and X_{2i} are the first and second coordinates of the i'th known 2D point in the neighborhood (14:394). As stated earlier in this chapter, any order drift can be modelled by kriging - simply modify k in equation [12]. If a polynomial drift is not observed during structural analysis, other types of drifts can be easily modelled as any type of function of the geometric coordinates.

3.5 Structural Analysis

Structural analysis is "the process of attempting to simultaneously find satisfactory representations of the semivariogram and drift expression." (14:245). This process also determines the optimal neighborhood for the estimation. The neighborhood is the sample values in the kriging linear sum that are all within a certain distance of the estimated value. I term neighborhood size the number of sample values in the kriging sum.

[3]Davis (14) and David (13) present the universal kriging equations in expanded matrix form.

The semivariogram, drift, and neighborhood all influence each other and characterize the notion of localized continuity within a sample volume. The goal of the process is to find a model semivariogram that models the spatial correlation of sample values within a local zone of influence (neighborhood).

Structural analysis is usually performed prior to kriging. Prior to understanding the process of structural analysis, the semivariogram must be defined.

Semivariogram Definition

The semivariogram is a graph and/or formula (14) that

> ... represents the spatial variability of data ... can be thought of as an average difference squared between data a given distance apart in a given direction... [and] provides a quantitative value for the range of influence of a sample in any direction (31).

The semivariogram is used in the kriging system of equations as an approximation to the co-variance between sample values. The co-variance measures the inter-dependence or correlation of random variable occurrences, whereas the semivariogram measures the spatial dependence of regionalized variable occurrences based on the distance from each other.

There are two major types of semivariograms used in kriging - experimental and model. The experimental semivariogram is used to estimate the variance of differences in the sample data. The experimental semivariogram is an estimator of the model semivariogram. Once an experimental semivariogram is computed, it is compared to known model semivariograms to select the closest match. The reason for using a model semivariogram is discussed in the section Models. The remainder of this section will describe how the experimental semivariogram is derived.

The semivariogram $\gamma(h)$ is often improperly called the variogram. It is termed the semivariogram because it is half the variogram $2\gamma(h)$ The variogram is:

$$2\gamma(h) = var(Z(p_i + h) - Z(p_i)) \qquad \forall p_i, \quad p_i + h \in N$$

where $2\gamma(h)$ is the variance between samples a distance of h apart from each other and N is the neighborhood of known sample points.

The semivariogram can be represented by a formula or a graph. Graphs depict the distance h on the abscissa and the semivariogram $\gamma(h)$ on the ordinate. The experimental or sample semivariogram (graph) is computed and plotted from the known sample points and values and is compared against known model semivariograms to determine the best fit (closest match). After a fit is made, model parameters are estimated. The semivariogram used in equations [8] and [11] is a model function, not experimental.

Davis states that the following equation can be used for estimating the semivariogram (experimental) for multiples of h when h is the same between data points (in other words the data is regular) :

$$\gamma_h^* = \frac{\sum\left[(X_i - X_{i+h}) - \frac{\sum(X_i - X_{i+h})}{n}\right]^2}{2n} \qquad [13]$$

The asterisk indicates this semivariogram is experimental or estimated from the sample values. This expression takes into account drift in the inner second term in the numerator,

$$\frac{\sum(X_i - X_{i+h})}{n} \quad [14]$$

Equations [13] and [14] are in (14).

Process of Structural Analysis

The goal of structural analysis is to determine a model semivariogram. To find this model semivariogram, an experimental semivariogram is first estimated from the known data values and compared to known model semivariograms to find a close match. However, before this is done, it must be determined if drift exists. The drift expression and the experimental semivariogram change based on the size of the neighborhood[4]. It would be best to estimate the semivariogram using the entire data set. However, it is too costly and often does little good since a distance is usually reached at which the affect of values on one another becomes neglible. Therefore, a maximum distance for the neighborhood is assumed initially to determine drift experimentally. This same distance is used in calculating the experimental semivariogram.

In the case of ordinary kriging, no drift exists so the experimental semivariogram calculated from the orginal sample values is sufficient to determine the spatial correlation of the samples. If drift exists, the situation is more complex because the semivariogram is not reliable statistically.

The model semivariogram must provide good statistical properties, like correlation between sample points based on spatial relationships. However, estimating the semivariogram of non-stationary regionalized variables may not have these kind of properties. According to Davis, stationary variables (regionalized variables with stationary means) force equation [14] to zero, which gives equation [13] a known statistical property - "the difference between the variance and the spatial autocovariance for the same distance." Normalizing the variables, i.e., mean zero and variance 1, provides an even better statistical property - the semivariogram becomes the "mirror image of the autocorrelation function" (14:242).

The main problem in the presence of global drift is that the experimental semivariogram is not reliable statistically. Recall that drift can exist in two forms, local and global. Local drift is accounted for in universal kriging. Therefore, the main task of structural analysis in this case is to compute a reliable experimental semivariogram in the case of global drift. Since stationary regionalized variables are considered reliable and residuals are stationary, then the residuals can be used to

[4]The drift coefficients as well as the order of the drift polynomial may differ.

compute the experimental semivariogram. To find the residuals, which are the drifts subtracted from the actual values, equation [9] must be solved.

The weights in equation [9] are determined in two possible ways. First, they can be estimated by the Lagrange Multipliers in the universal kriging system or they can be separately estimated by a regression technique like least-squares (13). If a kriging system is used, a known semivariogram can be assumed at the start (a "first approximation") to obtain estimates of the drift. If a least-squares approach is used, a first- or second-order polynomial is fit to the sample data to obtain estimates of the drift coefficients. In this case the d_l's in equation [9] are estimated as linear combinations of the data :

$$d_l^* = \sum_{i=1}^{n} w_{il} Z(p_i)$$

After estimates of the drift are obtained, they are then removed from the actual data to obtain residuals. The residuals are then used to estimate an experimental semivariogram. The experimental semivariogram is then compared to known models. If a poor fit results, the "first approximation" semivariogram, the neighborhood size, and/or even the order k of the polynomial drift equation [12] can be modified to obtain a closer fit. This recursive process is known as structural analysis. There is a strong "interrelation between neighborhood size, drift, and semivariogram for the residuals" (14).

Once a model fit is obtained, a kriging system is used with the model semivariogram to obtain estimates from the residuals. To get the final kriged estimates, the drift estimates are added to the estimates determined by kriging the residual regionalized variables.

In summary, the first step in structural analysis is to determine if global drift exists in the data set. This requires calculating a drift or sample mean for every sample value in the data set. If it exists, global drift must be removed by calculating residuals and then computing the experimental semivariogram from these residuals. Once a model semivariogram is chosen, values are estimated by kriging the residual data set with this model semivariogram. These kriged values are then added back to the drifts calculated at the beginning of the analysis. Or, if global drift does not exist, the data itself is used to determine the experimental semivariogram. If global drift exists or it does not, local drift can be assumed because if local drift does not exist, the local drift coefficients will just be zero. Global drift is not so easily accounted for because it involves a much larger neighborhood of sample values.

Models

Model semivariograms are used in the kriging system of equations rather than experimental semivariograms. Experimental semivariograms are not used because they do not provide results for distances other than those derived from the sample data. The term model refers to a known, continuous semivariogram (31) and (14:246). An experimental semivariogram like equation [13] is not used in equations [8] and [11] because the experimental semivariogram is computed for known discrete distances. Instead, a known model semivariogram computed for continuous distances is used. Since kriging estimates new values at possibly different distances than those between the original sample points, a continuous function is the most reasonable choice (14:246). Some of the better known continuous semivariogram models include the linear, spherical, and exponential (see figure 3.4).

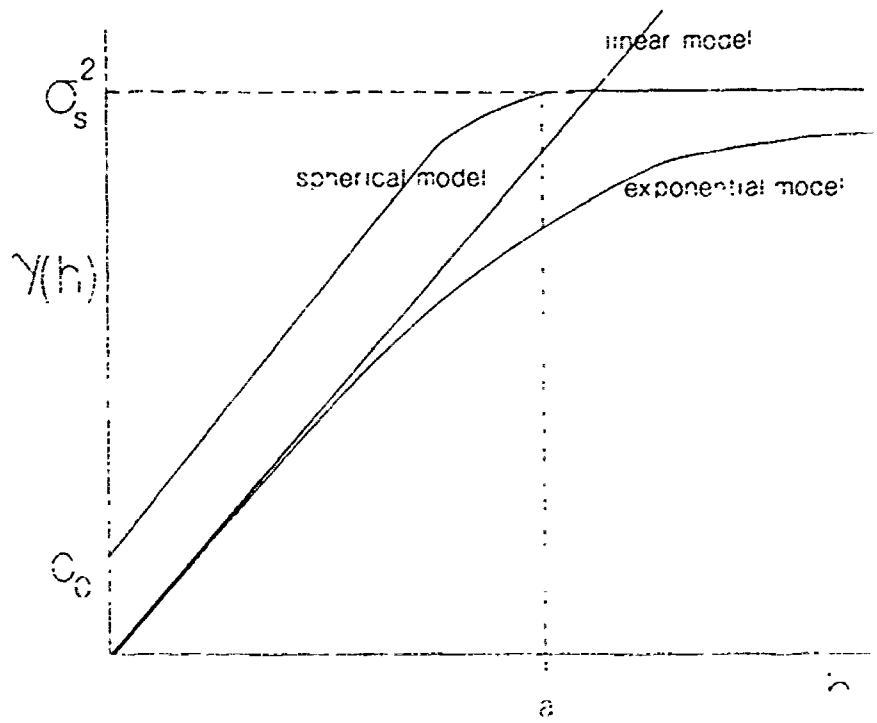


Figure 3.4. Example semivariograms for linear, spherical and exponentional models

Once a model is selected, parameters of the model must be estimated. According to Cressie (11:198), Zimmerman and Zimmerman claim that the "weighted

least squares approach usually performs well and never does poorly" at estimating parameters for a model semivariogram from the original data points.

Two terms are used to describe the models. Since the data is assumed to vary within some region, there usually exists some distance at which the data ceases to vary and has some constant variance. This constant variance is a flat line on a semivariogram known as the *sill*, σ_s^2. For models with a sill, the sill is the sample variance (6:7). The distance at which this constant variance is reached is termed the *range* a, or zone of influence. The range then determines the neighborhood of sample values used in the kriging equations because it represents the spatial zone in which sample values influence each other. Beyond this zone, the influence of a sample value on a kriged value is negligible. For a further description of sill and range see (10).

The following discussion of semivariogram models is based on (14:247), (31), and (13).

Linear Model. The following equation is used for this model:

$$\gamma(h) = ah + b$$

The parameters are the distance h and the slope of the line. This model does not have a sill, but sometimes the function is modified to create an artificial sill as follows:

$$\gamma(h) = ah + b \text{ for } h < a$$

$$\gamma(h) = \sigma_s^2 \text{ for } h \geq a$$

where a is the range. Davis suggests this approximation is good for "distances much less than the range" (14:247).

Spherical Model. The model equation is :

$$\gamma(h) = \begin{cases} C(\frac{3}{2}\frac{h}{a} - \frac{1}{2}\frac{h^3}{a^3}) + C_0 & \text{if } h < a \\ C + C_0 & \text{if } h \geq a \\ 0 & \text{if } h = 0 \end{cases}$$

The spherical model is most commonly used because it models most natural data very well. The relationship modelled between data values is that spatially closer data values have a higher influence on each other than do those further apart - until the range or limit of influence is reached. Once the range is reached, the values cease to have a significant affect on each other. The model parameters are the distance (h), the range (a), the sill ($C + C_0 = \sigma_s^2$), and the *nugget effect* (C_0).

The nugget effect measures micro-scale variations. It is the position on the $\gamma(h)$ axis where the semivariogram intersects, possibly causing a discontinuity at the origin if $C_0 \neq 0$ (35).

Exponential Model. The model equation is :

$$\gamma(h) = \sigma_s^2(1 - e^{-h/a})$$

The parameters are the distance (h), range (a) and sill (σ_s^2). This model is characterized by a semivariogram approaching the sill asymptotically. This indicates the data values always influence each other regardless of distance apart; however, values separated by distances beyond the range have much less influence on each other than those values separated by distances less than the range.

3.6 Isotropy/Anisotropy

Matheron (35:1249) noted three "qualitative characteristics" exhibited by regionalized variables. The first is support. Support consists of the shape, orientation, size, and spatial arrangement of the sample values. The second is that the sample values within a support show some form of continuity. The third, termed anisotropy, indicates the distance in the "zone of influence" (13:68) varies along different directions. Methods exist for treating different forms of anisotropy, but the discussion of the forms is beyond the scope of this research. For further information on this topic see (25) and (13).

3.7 Chapter Summary

This chapter has described some of the most basic forms of kriging. Structural analysis and kriging are complex, robust processes. It is optimal because the estimation variance is minimized, the estimation is unbiased, and the covariance approximation, the semivariogram, analyzes sample points based on their inter-dependence.

There are many current 3D medical imaging applications in use today that can use kriging to obtain more accurate estimates. In the next chapter, I will demonstrate how kriging can be used to estimate intra-cell scalar values in cell interpolation surface extraction and the volume pre-processing operation of slice interpolation.

IV. Cell Subdivision and Slice Interpolation Implementation

4.1 Introduction

I implemented two methods to explore estimation in 3D medical imaging. First, I implemented a cell subdivision method to investigate intra-cell scalar value estimation. I also implemented a method of estimating values between two medical data slices for the purpose of investigating estimation techniques in the volume preprocessing operation of slice interpolation.

This chapter begins by presenting an overview of cell subdivision and surface formation. Cell subdivision is discussed to explain how intra-cell points are derived. Once values are estimated at intra-cell points, the surface is formed. The process of surface formation is briefly discussed in this chapter. Then, the two estimation techniques, tricubic interpolation and kriging are discussed to show how they are used to estimate values at the intra-cell points derived by cell subdivision. The chapter ends by discussing my implementation of medical image slice interpolation. Appendix C provided further implementation details of the cell subdivision method.

4.2 Overview of Cell Subdivision and 3D Surface Formation

Presented in this section are the cell subdivision and 3D surface formation processing steps. Each step is addressed in turn. Before listing these steps, some terminology that helps in understanding the following explanations is discussed. Voxels are treated as point values, not volumes. I consider major cells the initial computational cells before subdivision (see figure 4.2). The cells created within a major cell by cell subdivision. I term minor cells. Major cells have the original voxel points and associated values as vertices. I consider minor cell vertices as minor-voxel points and values. Minor-voxel values are derived by intra-cell scalar value estimation. Finally, I consider the arrays within a major subdivided cell as mini-slices, since logically they represent input data slices. The reason for this will be explained in the discussion of steps 1 and 2.

The primary steps in the cell subdivision and surface formation algorithm are :

1. Read data slices into memory.

2. March major cells between slices.

3. Subdivide major cells into minor cells.

4. Estimate minor-voxel values and normals.

5. Apply marching cubes surface extraction within major cells to

6. Render triangular mesh with a polygonal based renderer. form surface.

Steps 1 and 2

These two steps are part of the marching cubes algorithm developed by Lorensen and Cline (34)[1]. Four slices of data are processed at a time. The marching cubes algorithm creates computational cells (cubes) between the two inner slices (see figure 4.1) and approximates the surface within each cell. My cell subdivision algorithm works similarly, except, surface formation is approximated in minor cells, not major cells[2].

Steps 3 and 4

When a major cell is processed, it is subdivided based on subdivision factors in each of the three component directions. For simplicity, assume component subdivision factors are equal. A subdivision factor of two will divide a major cell into eight minor cells (see figure 4.2). The 3D minor-voxel points are obtained by calculating division points along lines between major cell vertices based on the subdivision factors. A subdivision factor of two creates one division point along each line, a subdivision factor of three creates two division points along each line. etc., again assuming the same factor in all three directions.

The scalar values at these division points are estimated by one of the three estimation techniques - trilinear interpolation, tricubic interpolation, or kriging. Trilinear interpolation assumes the voxel values vary linearly within a cell in all three directions. This method assumes only the eight major cell vertices contribute to the estimation of intra-cell scalar values (minor-voxel values). Since trilinear interpolation is a standard method, details are not given here, but are in appendix H Tricubic interpolation is a parametric cubic polynomial interpolation method. This method

[1]During this effort, I implemented the marching cubes algorithm described by Lorensen and Cline (34). Details of this implementation are in appendices A and B. While developing the cell subdivision code I re-used as much of the marching cubes code as possible.

[2]My implementation can also perform the marching cubes algorithm on major cells.

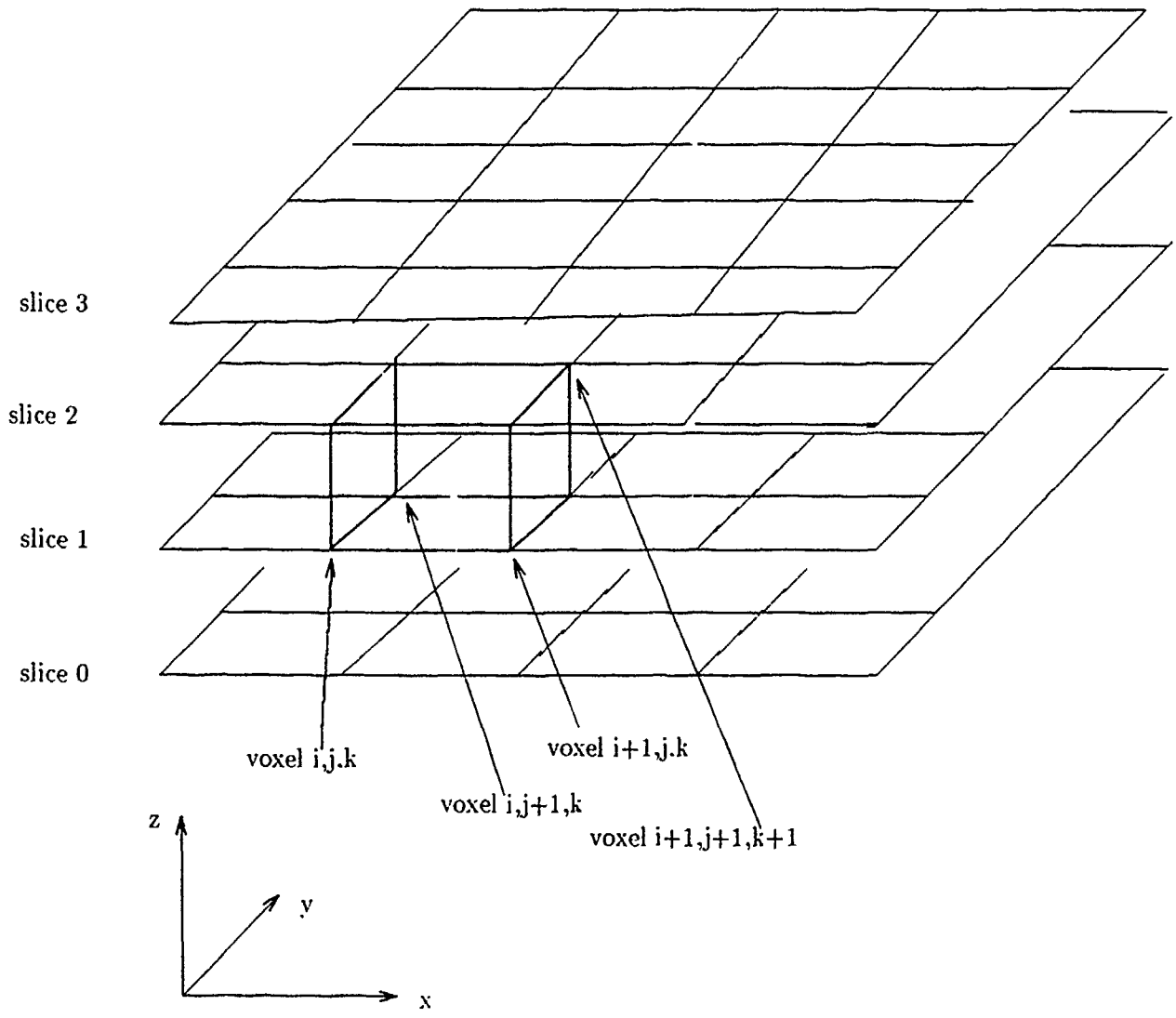
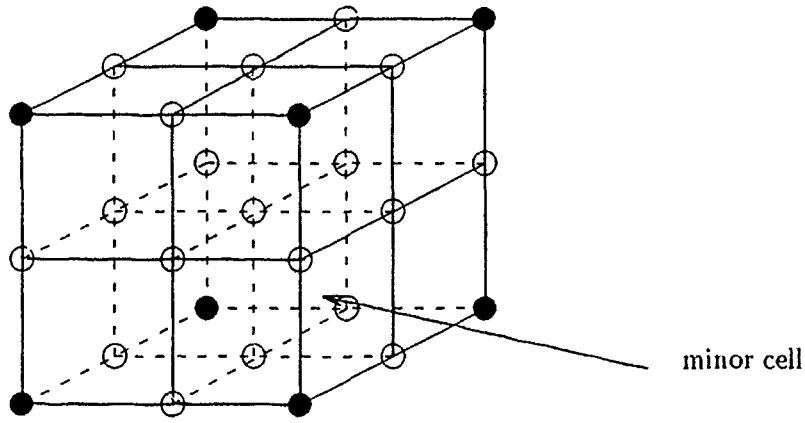


Figure 4.1. Computational cell (cube) marching between data slices



Subdivision factor = 2 in all dimensions

● Original voxels (major cell vertices)

○ Minor voxels (minor cell vertices)

Figure 4.2. Subdivided Major Cell

uses a much larger neighborhood of sample values to estimate minor-voxel values. It uses the surrounding 64 voxel values, where the major cell is centered within the next larger cube (see figure 4.3). Note, the 3D coordinates in the figure correspond to summation indices in the tricubic interpolation implementation discussed in the next section.

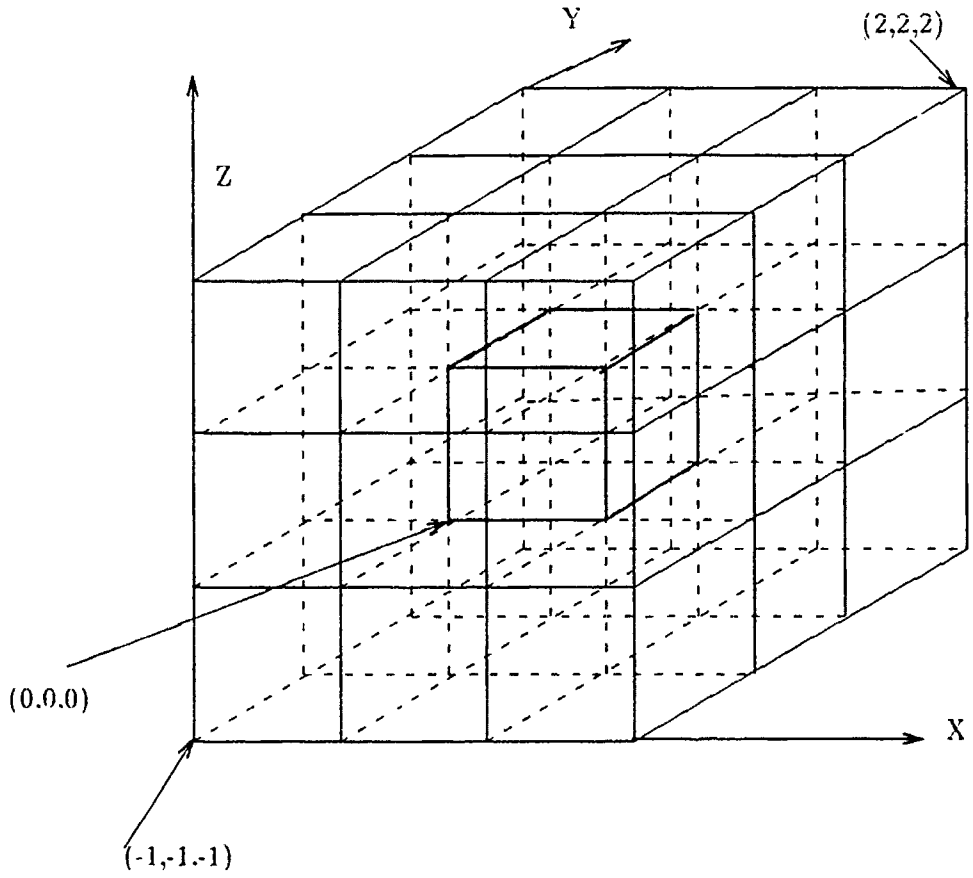


Figure 4.3. Major cell centered within surrounding cube

Tricubic interpolation assumes values vary cubically within a major cell. i.e., they fit a 3D curved surface within the major cell. Trilinear interpolation and tricubic interpolation are termed deterministic because the procedures do not account for error. Kriging on the other hand estimates values based on the statistics of the data and not only accounts for error, but minimizes it. It estimates values using a weighted linear combination of nearby known sample values (voxel values). The

weights are determined by conditions that insure unbiased sampling and minimum estimation error variance. The latter condition requires that co-variances between sample values be computed. These co-variances are approximated by a technique that computes the average difference squared (in distance) between data samples. This causes sample values closer to the value being estimated to have more influence in the estimation than sample values farther away. Therefore, kriging is really a distance weighted estimation function. It does not assume linear,quadratic, or any form of variation, although it can be tailored to do so. In fact, I tailor kriging to behave like both a tricubic and a trilinear interpolator. Details of this tailoring and tricubic interpolation are discussed in the next two major sections of this chapter.

The purposes of cell subdivision are to 1) resolve ambiguity in cells and 2) provide a better surface approximation. Recall a cell is ambiguous if more than one topology can be chosen to represent the surface within the cell. Cell subdivision guarantees that major cells will be disambiguated only because they are being subdivided. That is, once subdivided, major cells are no longer treated computationally, hence they are disambiguated. The minor cells are now the computational units. The problem with cell subdivision as a disambiguation method is that minor cells may still be ambiguous. If minor cells are ambiguous, Wilhelms and Gelder (50) apply the facial averaging technique discussed in chapter two to disambiguate these. I do not do this here, because my goal is to investigate the use of kriging to estimate intra-cell values. Besides disambiguating ambiguous major cells, cell subdivision also provides a better surface approximation within each major cell. This is because the surface is now being detected at a finer sampling – although many of the values are not original *samples*, but rather estimated values. How well the extracted surface corresponds to the actual surface depends mainly on the accuracy of the estimation function employed. To better understand how subdivision can not only disambiguate most minor cells, but also form a smoother surface, see appendix D.

Step 5

There are two marching cubes implementations used in the cell subdivision process. The first is the outermost loop. In this loop, data slices are read into memory and major cells are formed. At this point, a vanilla marching cubes implementation can be selected[3]. If the vanilla marching cubes implementation is not selected, cell

[3]Vanilla indicates no cell subdivision, disambiguation, or enhancements to the original algorithm

subdivision is performed. The second marching cubes implementation occurs within each nonempty major cell. In this case, mini-slices formed by the subdivision are treated as if they are actual data slices. This *minor* marching cubes implementation treats minor cells as "cubes." Within each minor cell, triangle vertices and normals are computed to represent the portion of the surface passing through each cell. Details of triangle formation and normal computation are in appendices A,B and C.

Step 6

The triangular mesh is rendered using Phong reflectance and Phong shading. Details of the renderer used are in Appendix F.

A more detailed discussion of cell subdivision is in appendix C.

4.3 Tricubic Interpolation

The tricubic interpolation estimation technique assumes that estimated data within a major cell fits a cubic surface within the cell. In a two-dimensional sense, this is like fitting points to a curved line between two known points (see figure 4.4)

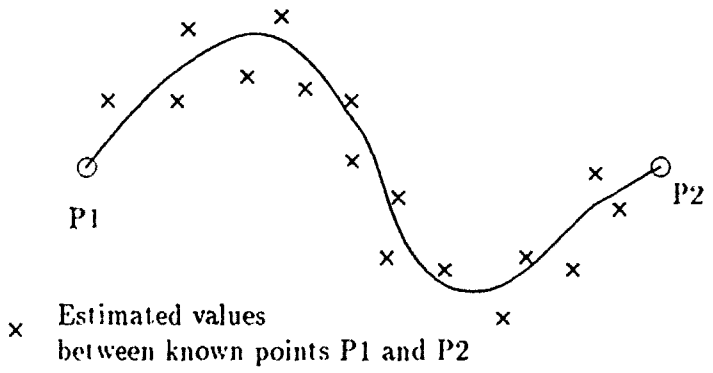


Figure 4.4. Example of fitting points to a curved line in 2D

The tricubic polynomial interpolation method implemented is the same one implemented by Wilhelms and Gelder (50:15). The parametric cubic polynomial interpolation method they and I use is very similar to the hermite form of a cubic polynomial curve, with constraints being two endpoints and tangent vectors at the endpoints (20). Two differences exist between the classical hermite form of a cubic polynomial and the one used here. The classical form is used to generate points

on a surface. Here we are estimating values that lie on or near a surface in 3D, not the points that generate the surface. Also the tangent vector constraints are determined differently, based on the volume data. The following equation represents the formulation of the values in one dimension:

$$F(u) = au^3 + bu^2 + cu + d = \mathbf{U} \cdot \mathbf{C_x} = \mathbf{U} \cdot \mathbf{M_H} \cdot \mathbf{G_H} = [u^3 \;\; u^2 \;\; u \;\; 1]\mathbf{M_H} \cdot \mathbf{G_H}$$

where $0 \leq u \leq 1$, $\mathbf{M_H}$ and $\mathbf{G_{H_x}}$ are the Hermite basis matrix and geometry vector.

Normally, the tangent vector constraints are determined by differentiating the **U** vector and solving at u = 0 and u = 1, the endpoints of the curve segment. I, as Wilhelms and Gelder do, modify these constraints by assuming the derivative of a value f_i at a point i in one dimension is approximately the central difference $f_i' = \frac{1}{2}(f_{i+1} - f_{i-1})$. The blending functions for a univariate curve $F(u)$ are determined by solving a system of equations including the constraints $F(0) = f_0$, $F'(0) = f_0'$, $F(1) = f_1$, and $F'(1) = f_1'$. The solution of the system results in the following equation for an estimated value in one dimension :

$$F(u) = \sum_{-1}^{2} f_i B_i(u)$$

where $B_i(u)$ are the blending functions :

$B_{-1}(u) = \frac{1}{2}(-u^3 + 2u^2 - u)$
$B_0(u) = \frac{1}{2}(3u^3 - 5u^2 + 2)$
$B_1(u) = \frac{1}{2}(-3u^3 + 4u^2 + u)$
$B_2(u) = \frac{1}{2}(u^3 - u^2)$

The indexing scheme (-1 to 2) is used to correspond to the position of a cell's eight vertices in relation to the surrounding voxels. The point (0,0,0) is the bottom front left vertex of a major cell (see figure 4.3)[4].

The tricubic function is determined by applying the equation in all three components and is given by

$$F(u,v,w) = \sum_{k=-1}^{2} \sum_{j=-1}^{2} \sum_{i=-1}^{2} f_{i,j,k} B_i(u) B_j(v) B_k(w)$$

[4]Although not shown in the figure, the major cell centered in the larger surrounding cube is subdivided

$F(u, v, w)$ estimates minor-voxel values at the intra-cell points derived by cell subdivision. u, v and w range between 0 and 1. u is equal to the fraction of the distance between the major cell vertices in the x direction. Similarly, v is the fraction of the distance between the major cell vertices in the y direction, and w corresponds in like manner in the z direction.

This function constrains the surface to the computational cell because the two endpoint values in the formulation for each dimension are cell vertex values. The term tri stems from the three parameters. For the bicubic case, Watt (49) describes the surface formulation as a cartesian product of two curves. In the tricubic case, the surface formulation is the cartestian product of three curves.

4.4 Kriging Estimation

This section discusses the kriging estimation technique used to estimate minor-voxel values. First, an overview of the technique is presented. Following that, specific implementation conditions are listed and discussed. Lastly, the kriging estimation procedure implemented in this research is presented.

The kriging code developed during this effort was written by Capt Chris Brodkin (3) and modified for use in intra-cell scalar value estimation. It is object-oriented code written in C++.

4.4.1 Overview of Kriging Technique Kriging estimates minor-voxel values by a weighted linear sum of nearby original voxel values. The conditions of unbiasedness and minimum estimation error variance yield a system of solvable equations in the familar matrix equation form $X = A^{-1}B$, where the solution X matrix contains the desired weights. However, before this matrix equation can be solved, the model semivariogram in the A and B matrices must be determined.

The goal of this research is simply to demonstrate that kriging can be used to estimate intra-cell scalar values and that it is flexible. To ensure that kriging can estimate points similar to the other methods, I tailored it to behave like tricubic and trilinear interpolations. To achieve this, structural analysis is not necessary. However, structural analysis is necessary if the goal is to calculate the best estimates. To tailor kriging to behave like tricubic and trilinear interpolations, some assumptions were made :

- Global drift does not exist.
- Local drift may exist.
- The model semivariogram is known.
- The neighborhood size is known, but alterable.
- The data is isotropic.

4.4.2 Global and Local Drift Drift is the phenomena that occurs when sample means vary from point to point. A sample mean is derived by choosing some neighborhood of values surrounding a known sample point and calculating a weighted average. If these weighted averages (sample means) differ from point to point, then drift exists. If they are the same, then the sample means are constant.

Drift can occur in two ways. It can occur within local regions (local drift) and/or throughout the entire data set (global drift). If global drift exists, it is removed by calculating residuals from the sample means. A residual is calculated by subracting the estimated drift from the sample value at a known point. Residual data is considered to be nonstationary, i.e., there is no drift. Since kriging only works with nonstationary data, residuals are then kriged to estimate values. These values are then added to the sample means to get the final estimates.

Drift can also occur just within the neighborhood of sample values being used to determine the linear sum. This local drift can be accounted for in the universal kriging system of equations.

Global drift is assumed not to exist in this implementation to simplify the process. Local drift is incorporated into the universal system by the polynomial form of the drift expression. Even if global drift exists, it should not have a large effect if local drift is accounted for.

The three-dimensional local linear drift expression implemented for this effort in terms of the geometric coordinates x_i, y_i, z_i is

$$m(p_i) = m(x_i, y_i, z_i) = d_0 + d_1 x_i + d_2 y_i + d_3 z_i$$

and the three-dimensional local quadratic drift expression is

$$m(p_i) = d_0 + d_1 x_i + d_2 y_i + d_3 z_i + d_4 x_i^2 + d_5 y_i^2 + d_6 z_i^2 + d_7 x_i y_i + d_8 x_i z_i + d_9 y_i z_i$$

4.4.3 The Assumed Model Semivariogram The model semivariogram is a continuous function that takes the distance between two sample points as a parameter. It is used in the kriging system of equations as an approximation to the co-variance between sample values to give geometric meaning to the values instead of probabilistic meaning as in classical statistics. The co-variance measures the inter-dependence or correlation of sample values, whereas the semivariogram measures the spatial dependence of sample values based on distance. Two types of semivariograms exist. These are the experimental and the model. The experimental semivariogram is a discrete function derived from the data set prior to kriging. It is computed as an average difference squared between data points. The experimental semivariogram is then compared to continuous, known model semivariograms to find the best match. Model semivariograms are actually used in the kriging equations because distances other than those found in the data set might be used to estimate new points.

To tailor kriging to behave like tricubic interpolation, an experimental semivariogram was not computed and the model semivariogram was assumed to be a cubic function. This assumed semivariogram model is also called a drift model, (31), see figure 4.5.

This model indicates a polynomial drift exists in the data, which is certainly the assumption made in the tricubic interpolation method discussed above. The function used is simply

$$\gamma(h) = abs(h^3)$$

This model semivariogram coupled with the drift expressions above parallels the tricubic method. The tricubic method is derived from parametric cubic functions.

4.4.4 Neighborhood Size Recall from chapter three that neighborhood size is the number of sample values in the kriging estimation equation. Neighborhood size is usually determined during a structural analysis of the data (structural analysis was discussed in chapter three). Instead of determining the neighborhood by a structural analysis of the data, I assumed different sizes. First, I assumed the same neighborhood size as that used in tricubic interpolation, 64. This neighborhood size is the 56 surrounding points, including the cell's eight points, for a total of 64 sample values in the kriging system (see figure 4.3). The neighborhood sizes 8, 16, and 32 were also investigated for both cell subdivision intra-cell scalar value estimation and slice interpolation. These sizes were chosen because they were easily implemented

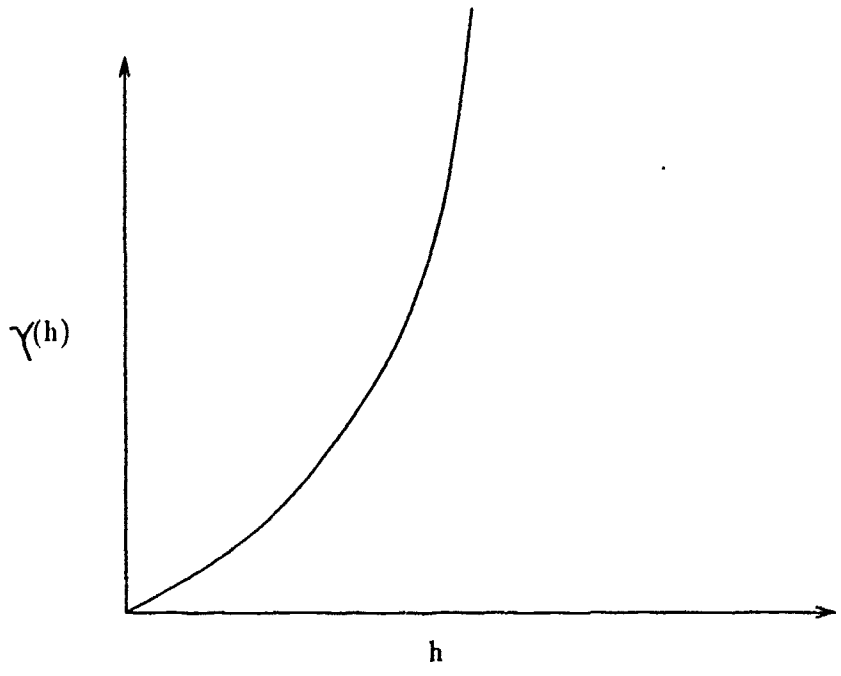


Figure 4.5. Drift model variogram

from the cell geometry. They were used to demonstrate the effect of kriging with different neighborhood sizes.

4.4.5 Estimation Procedure The goal of kriging restated in terms of this particular problem is to estimate $\hat{Z}$

$$\hat{Z}(p) = \sum_{i=1}^{n} w_i Z_i(p_i)$$

where the p_i and Z_i's are the surrounding voxel points and values and p is the 3D point where the value $\hat{Z}$ is being estimated. n = 8, 16, 32 or 64 in this equation and all the following ones for my implementation.

Both the ordinary and universal forms of kriging were implemented. Recall the ordinary system is

$$\sum_{j=1}^{n} w_j \gamma_{ij} + \eta = \gamma_{ip} \qquad (i = 1, ..., n)$$

$$\sum_{j=1}^{n} w_j = 1$$

This system of equations is represented in the matrix form $AX = B$, where the X column vector contains the kriging weights.

$$\begin{bmatrix} \gamma_{11} & \gamma_{12} & \cdots & \gamma_{1n} & 1 \\ \gamma_{21} & \gamma_{22} & \cdots & \gamma_{2n} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \gamma_{n1} & \gamma_{n2} & \cdots & \gamma_{nn} & 1 \\ 1 & 1 & \ldots & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ \eta \end{bmatrix} = \begin{bmatrix} \gamma_{1p} \\ \gamma_{2p} \\ \vdots \\ \gamma_{np} \\ 1 \end{bmatrix}$$

The solution to this matrix equation is $X = A^{-1}B$, where A^{-1} is the inverse matrix of A.

For the universal system, the equations change depending on the assumed local drift, $f^l(p_i)$. The system is

$$\sum_{j=1}^{n} w_j \gamma_{ij} + \sum_{l=0}^{k} \eta_l f^l(p_i) = \gamma_{ip} \qquad (i = 1, ..., n)$$

$$\sum_{i=1}^{n} w_i f^l(p_i) = f^l(p) \qquad (l = 0, 1, ..., k)$$

The following equations contain quadratic drift. Linear drift is obtained by removing all the quadratic terms. The universal kriging system with local quadratic drift implemented in this research is

$$\sum_{j=1}^{n} w_j \gamma_{ij} + \eta_0 + \eta_1 x_i + \eta_2 y_i + \eta_3 z_i + \eta_4 x_i^2 + \eta_5 y_i^2 + \eta_6 z_i^2 + \eta_7 x_i y_i + \eta_8 x_i z_i + \eta_9 y_i z_i = \gamma_{ip}$$

$(i = 1, ..., n)$

$$\sum_{i=1}^{n} w_i = 1 \quad (l = 0)$$

$$\sum_{i=1}^{n} w_i x_i = x_p \qquad (l = 1)$$

$$\sum_{i=1}^{n} w_i y_i = y_p \qquad (l = 2)$$

$$\sum_{i=1}^{n} w_i z_i = z_p \qquad (l = 3)$$

$$\sum_{i=1}^{n} w_i x_i^2 = x_p^2 \qquad (l = 4)$$

$$\sum_{i=1}^{n} w_i y_i^2 = y_p^2 \qquad (l = 5)$$

$$\sum_{i=1}^{n} w_i z_i^2 = z_p^2 \qquad (l = 6)$$

$$\sum_{i=1}^{n} w_i x_i y_i = x_p y_p \qquad (l = 7)$$

$$\sum_{i=1}^{n} w_i x_i z_i = x_p z_p \qquad (l = 8)$$

$$\sum_{i=1}^{n} w_i y_i z_i = y_p z_p \qquad (l = 9)$$

x_i, y_i, and z_i are the components of the ith 3D sample point.

The following is the matrix form of this system. $AX = B$.

$$
\begin{bmatrix}
\gamma_{11} & \gamma_{12} & \cdots & \gamma_{1n} & 1 & x_1 & y_1 & z_1 & x_1^2 & y_1^2 & z_1^2 & x_1y_1 & x_1z_1 & y_1z_1 \\
\gamma_{21} & \gamma_{22} & \cdots & \gamma_{2n} & 1 & x_2 & y_2 & z_2 & x_2^2 & y_2^2 & z_2^2 & x_2y_2 & x_2z_2 & y_2z_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\gamma_{n1} & \gamma_{n2} & \cdots & \gamma_{nn} & 1 & x_n & y_n & z_n & x_n^2 & y_n^2 & z_n^2 & x_ny_n & x_nz_n & y_nz_n \\
1 & 1 & \cdots & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_1 & x_2 & \cdots & x_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
y_1 & y_2 & \cdots & y_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
z_1 & z_2 & \cdots & z_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_1^2 & x_2^2 & \cdots & x_n^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
y_1^2 & y_2^2 & \cdots & y_n^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
z_1^2 & z_2^2 & \cdots & z_n^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_1y_1 & x_2y_2 & \cdots & x_ny_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_1z_1 & x_2z_2 & \cdots & x_nz_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
y_1z_1 & y_2z_2 & \cdots & y_nz_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ \eta_0 \\ \eta_1 \\ \eta_2 \\ \eta_3 \\ \eta_4 \\ \eta_5 \\ \eta_6 \\ \eta_7 \\ \eta_8 \\ \eta_9 \end{bmatrix}
=
\begin{bmatrix} \gamma_{1p} \\ \gamma_{2p} \\ \vdots \\ \gamma_{np} \\ 1 \\ x_p \\ y_p \\ z_p \\ x_p^2 \\ y_p^2 \\ z_p^2 \\ x_py_p \\ x_pz_p \\ y_pz_p \end{bmatrix}
$$

The A matrix is inverted to solve for the X column vector of unknown weights and Lagrange multipliers. Recall the semivariogram γ_{mn} is the same as $\gamma(h_{mn})$, where h_{mn} is the distance from point m to point n. The η_l's are the $k+1$ Lagrange multipliers where $k = 9$ and (x_p, y_p, z_p) is the point where the value is being estimated.

4.5 Slice Interpolation

This section describes how scalar values are estimated between medical slices for the volume pre-processing operation of slice interpolation. To accomplish this task I re-used as much of the cell subdivision code as possible. Computational cells are used in the estimation process. The estimation is done only along one cell edge because the purpose here is to interpolate only in the 'Z' direction. Recall the 'Z' direction is the direction that the data slices are stacked. Four different neighborhood sizes were used - 16, 32 and 64. The slice interpolation algorithm is :

1. Read in four medical data slices.
2. March cell (cube) between the two inner data slices.
3. Estimate scalar values for new slice(s) along cell edges.
4. Create gray scale image from estimates.

Four slices are needed at a time because the tricubic interpolation requires a neighborhood size of 64. Computational cells on the boundary of the data are treated specially. Since they do not have access to the larger neighborhood of sample values, I compute a linear interpolation along boundary cell edges. This should have no effect on the final image because the boundaries of images typically do not contain any significant data.

My implementation of linear interpolation is not presented because it is a standard method. The important point about linear interpolation is that it assumes only a linear variation. The tricubic interpolation and kriging estimation methods used here are the same ones described in the previous sections.

The next chapter presents the results of implementing the methods discussed in this chapter and in the appendices. The appendices contain descriptions of the methods used to accomplish the other tasks outlined in chapter one, yet not mentioned in this chapter.

V. Results

This chapter presents the results obtained from implementing the estimation methods discussed in the previous chapter. The results are divided into three major areas:

- Artificial Volume – Cell interpolation surface extraction and intra-cell scalar value estimation in an artificial volume.
- Medical Image Slice Interpolation.
- Medical Volume – Cell interpolation surface extraction and intra-cell scalar value estimation in a medical volume.

These results demonstrate that kriging subsumes the deterministic methods investigated. I modify the kriging system by changing the neighborhood size and local drift assumption. By doing this, I show that kriging can be tailored to behave like the other deterministic methods, i.e. it is flexible, and in some cases produce images that look better than the other methods[1].

Medical image slice interpolation is discussed prior to medical volumes because some analysis was done on a subset of the medical volume slices.

Explanation of pictures and tables

The labels in the images in the pictures presented in this chapter are file names that were chosen to be descriptive. For example, f2tricubic3.rle indicates that the image represents the f2 function, derived by cell subdivision with tricubic interpolation at a subdivision factor of 3. rle indicates the Utah run length encoded format. Similarly, f2krige3nodrift.rle indicates the f2 function derived by cell subdivision with kriging estimation assuming no local drift at a subdivision factor of 3. Also, f2krigenh32linear indicates the f2 function derived by cell subdivision with kriging estimation assuming local linear drift at an assumed subdivision factor. I will leave off the extension rle or rle.Z (.Z is compressed format) for brevity.

[1] When I discuss image accuracy or quality, this is my opinion about the visual appearance of the images.

The table headings are as follows. In the pictures derived by cell subdivision, the number of triangles and non-empty minor cells are listed. These are denoted as column headers "# TRI" and "# NEM" respectively. All other columns discussed are applicable to all the tables. The first is "Images". This header indicates one particular image in a picture. The header "values compared with" indicates another image compared with the image listed under the column "Images". The header "largest est value diff" presents the largest estimated value difference between the two compared images. This entry provides knowledge of the extreme deviation of sample values derived by two different estimations. As David states, "the most natural way to compare two values ... is to consider their difference." I actually take the absolute value for this column. David also states the average difference is an important measurement as well to understand the dissimilarity between values (13). This measurement is presented in the column under the heading "avg diff of values". The average difference of the values is derived by the formula

$$\frac{\sum_{i=1}^{n} abs(image1\_value_i - image2\_value_i)}{n},$$

where n is the number of values estimated, $image1\_value_i$ is the i'th value estimated in the generation of the image under the column "Image" and $image2\_value_i$ is the i'th value estimated in the generation of the image under the column "values compared with". Finally, the last column in all the tables indicates the average difference percentage of the total scalar value range. For example, the scalar value range for the f2 function in an artificial volume is 136.0. Also, the scalar value range for the slice interpolation tests is 256. 0 through 255.

5.1 Artificial volume

The main purpose of this section is to show that using kriging to estimate intra-cell scalar values resolves ambiguity in cells and that kriging is very flexible compared to the other two interpolation methods. By flexibility I mean that I can change parameters in the process that alter how kriging estimates. This cannot be done with the other techniques. Both trilinear and tricubic assume the neighborhood and local data variation. The advantage of kriging is that it can be tailored to behave like either of the other two interpolation methods, or it can be tailored to analyze any size neighborhood and assume other local data variations besides linear and cubic. First presented are images comparing kriging to the other two interpolation techniques, fixing kriging to behave like tricubic. Then, both the local drift and

the neighborhood size are changed to show how flexible kriging is and demonstrate how important these two factors are. Many other factors can be modified to make kriging more robust and more accurate, but the purpose of this research is to just demonstrate its usefulness and applicability.

An artificial volume is created by calculating scalar values at voxel points. The artificial volumes represent continuous 3D iso-surfaces derived from mathematical functions. The details of this process are in an appendix. The mathematical function used is a hyperboloid with known ambiguous cases. The function, which Wilhelms and Gelder called F_2 (50), is

$$F_2(x, y, z) = 4(y-1)^2 + 2(x-z)^2 - 2(x+z-3)^2 + 1$$

5.1.1 Neighborhood size 64, subdivision factor = 4, local drift assumption differs. Since the number of ways to compare images is quite large – taking into account neighborhood size, drift, and subdivision factors – I began by assuming a neighborhood size of 64 and a subdivision factor of four. Then I compared the estimated values derived from universal kriging (with local and quadratic drift) to the estimated values derived from ordinary kriging. The values were exactly the same in all three cases. This indicates that at least for this data set and a neighborhood size of 64, there is no local drift. I explored quadratic local drift in nearly all the cases presented in this chapter, but found only minor differences between these images and those assuming local linear drift. Therefore to reduce the number of images in the study, I only present images derived by kriging that assume local linear drift or no local drift.

5.1.2 Neighborhood size 64, subdivision factors differ, no local drift assumed. Next, I ran the program with the three different estimation methods at the subdivision factors two, three, four and five; still maintaining the kriging neighborhood size at 64 and assuming no local drift. The results can be seen in figures 5.1 – 5.7. All the cell subdivision methods - trilinear and tricubic interpolation and kriging - removed the ambiguous cells for all subdivision factors. In other words, after cell subdivision and intra-cell scalar value estimation, no ambiguous cases were found. The subdivision factor as well as the estimation function significantly alters how the surface is represented. The higher the subdivision factor, the smoother the surface representation. Also, tricubic interpolation and kriging tailored to behave like tricu-

bic interpolation both estimate values that produce smoother surface representations of the hyperboloid.

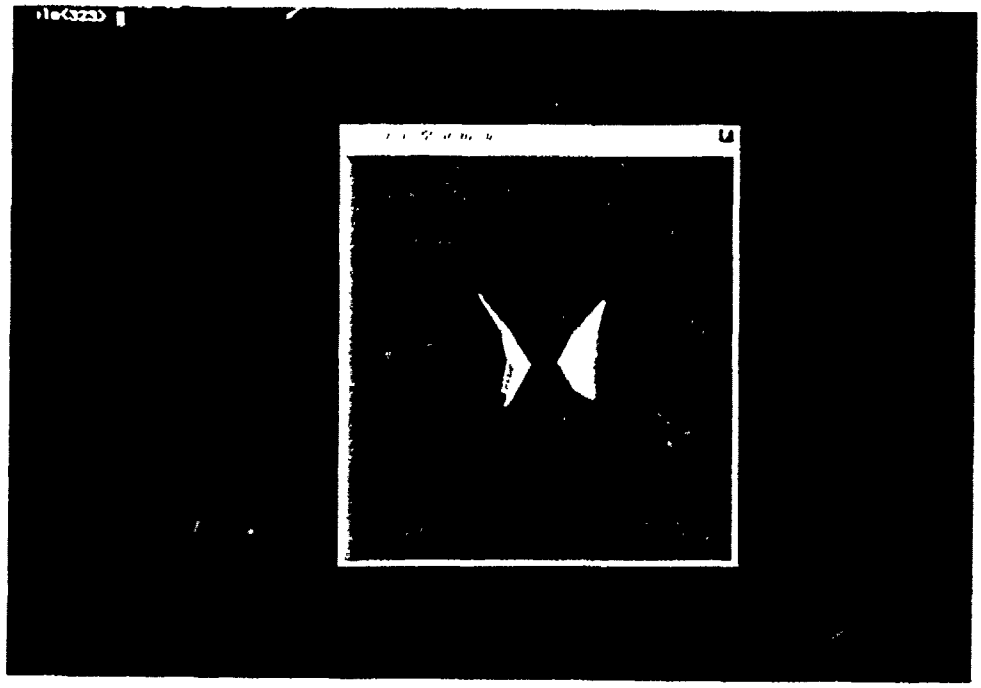


Figure 5.1. Vanilla marching cubes extraction of hyperboloid surface

Note the images obtained by using tricubic interpolation and kriging for intra-cell scalar value estimation look almost identical. This is verified by a comparison of the estimated values between the two methods. For example, at subdivision factor 4, values estimated by tricubic interpolation and kriging differ by an average of .0113 (range of scalar values is -49.0 to 87.0). See table 5.1 for a detailed comparison of the estimation techniques. TRI indicates triangles and NEM means Nonempty Minor Cells.

Several results are presented in this table. By analyzing this table and examining the images it is obvious that the kriging assumptions model tricubic interpolation very closely. For subdivision factors 2,3, and 4, the number of triangles as well as the number of nonempty minor cells are exactly the same for the tricubic and kriging estimation techniques. This indicates that the estimated values are very similar. To further show this, I compared values estimated as shown in the table, noting the largest difference in two values and the average difference of all the estimated values. These results confirm that the kriging assumptions made (61 neighborhood, no drift,

Table 5.1. Comparison of values estimated by trilinear and tricubic interpolation and kriging (with a 64 neighborhood, no local drift)

Image	# TRI	# NEM	values compared with	largest est value diff	avg diff of values	avg diff / 136 %
f2vanmc	44	-				
f2trilinear2	140	70				
f2tricubic2	140	70	f2krige2	.1363	.0031	.0023
f2krige2	140	70				
f2trilinear3	332	166				
f2tricubic3	348	174	f2krige3	.1211	.0069	.0050
f2krige3	348	174				
f2trilinear4	560	280				
f2tricubic4	576	288	f2krige4	.1331	.0113	.0083
f2krige4	576	288				
f2trilinear5	888	444				
f2tricubic5	920	460	f2krige5	.1316	.0160	.0118
f2krige5	912	456				

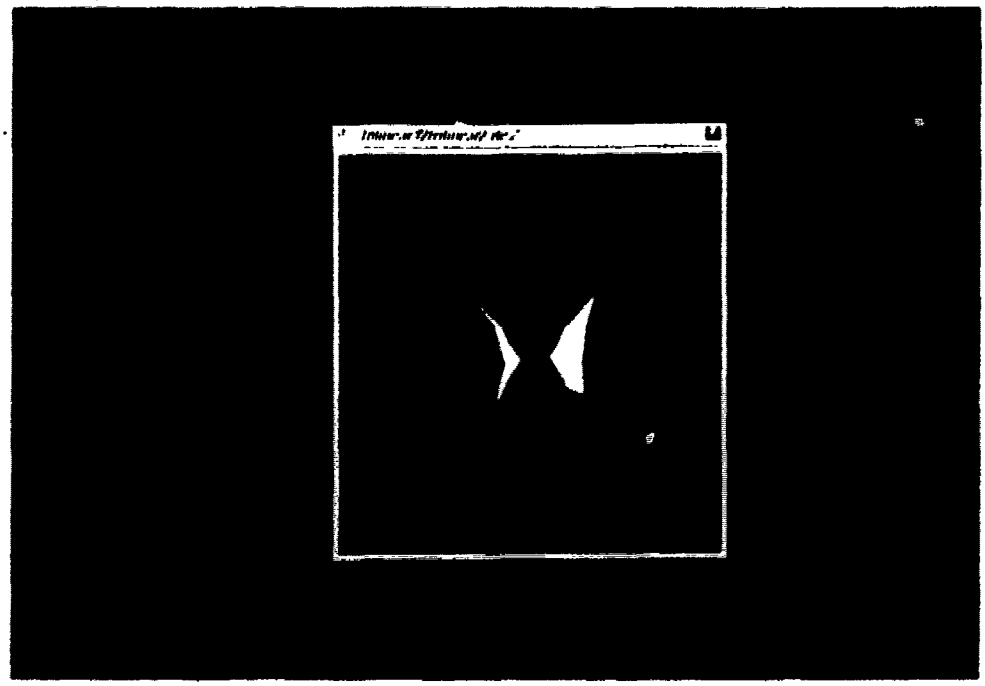


Figure 5.2. Cell subdivision.factor 2. with trilinear interpolation estimating minor-voxel values and marching cubes extraction of hyperboloid surface from mini-slices

h^3 model semivariogram. and isotropic data) model tricubic interpolation extremely well. However. as the next image shows. a neighborhood size of 64 is not necessary for kriging to estimate values close to the supposed accurate values of tricubic interpolation. Reducing the number of sample values directly affects execution time because it reduces the size of the matrix being inverted in the kriging system of equations.

5.1.3 Neighborhood size differs. Subdivision factor = 2. local drift assumption differs. The next set of images in this section demonstrates the flexibility of kriging (figures 5.9 - 5.13). In these sets of images. the subdivision factor is two, chosen arbitrarily. I altered both the neighborhood size and the local drift. In all the images except those with a neighborhood of 32. the images with no drift are significantly different than those with local linear drift. It appears that with smaller neighborhoods. assuming local drift prevents the inaccuracies seen by the holes.

The neighborhoods selected are as follows (see figure 5.8). A 32 neighborhood starts with the original 64 values surrounding the computational cell and ignores the

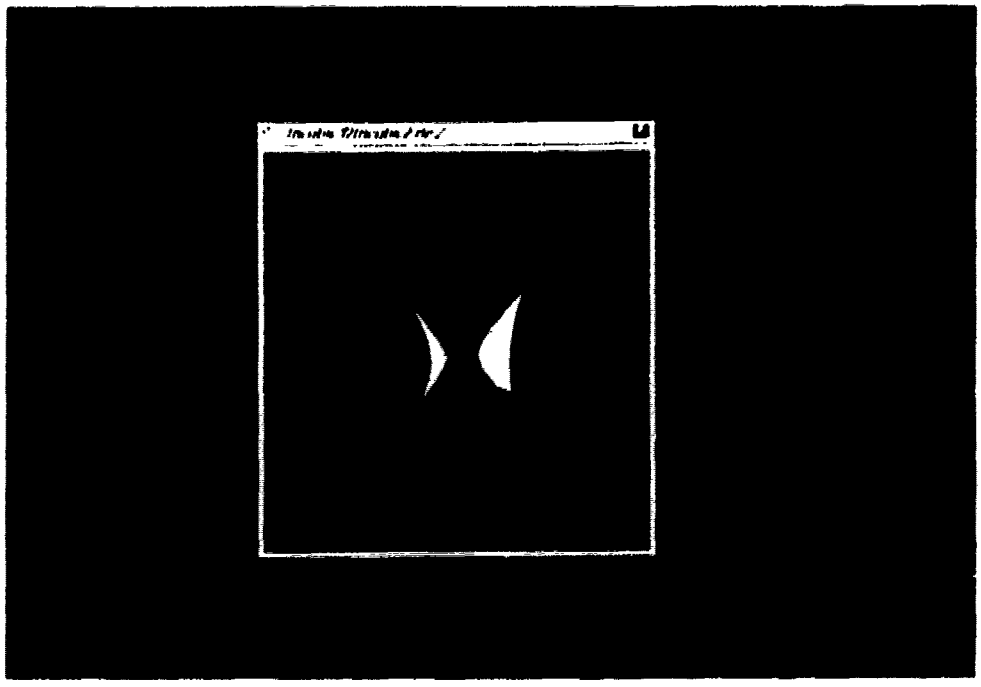


Figure 5.3. Cell subdivision, factor 2, with tricubic interpolation estimating minor-voxel values and marching cubes extraction of hyperboloid surface from mini-slices

values in the 2 slices above and below the computational cell. A neighborhood of 16 in the 'X' direction, denoted nh16x in the figures, consists of the eight computational cell vertices and four on either side of the cell in the 'X' direction. 16 'Y' and 16 'Z' neighborhoods are derived in a similar fashion. These particular neighborhoods were chosen because of their direct correspondence to the computational cell and the indexing scheme already used for tricubic interpolation and kriging in a 64 neighborhood.

Images depicting differences were created by the Utah RLE library tool rlecomp with the diff operator. This tool performs the logical set difference operation between pixel values in the two images.

Table 5.2 depicts differences in estimated values for the last sets of images. Notice that kriging with a neighborhood of 32 sample values and no local drift matches tricubic interpolation almost as closely as kriging with a neighborhood of 64 sample values does. I compared local linear drift kriging images to tricubic and linear because local linear drift kriging produced better images. As soon as the kriging neighborhood size reduces to 16, except in the case of nh16y, the kriging

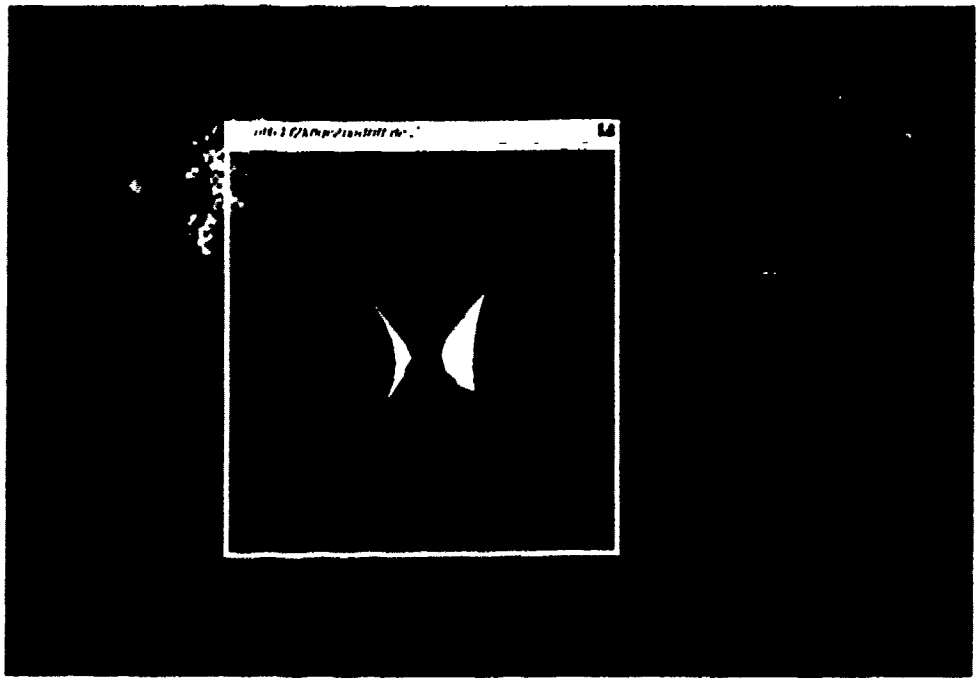


Figure 5.4. Cell subdivision, factor 2, with kriging estimating minor-voxel values and marching cubes extraction of hyperboloid surface from mini-slices. Kriging uses a neighborhood of 64 sample values and assumes no local drift.

Table 5.2. Comparison of values estimated by different kriging forms for cell interpolation surface extraction of a hyperboloid surface in an artificial volume

Image	values compared with	largest est value diff	avg diff of values	avg diff —— 136 %
f2krigenh32nodrift	f2krigenh32linear	.0310	.0004	.0003
f2krigenh32nodrift	f2tricubic2	.5066	.0098	.0072
f2krigenh16xnodrift	f2krigenh16xlinear	.1922	.0023	.0017
f2krigenh16xlinear	f2tricubic2	31.7500	1.2602	.9266
f2krigenh16xlinear	f2trilinear2	.0646	.0006	.0004
f2krigenh16ynodrift	f2krigenh16ylinear	.1617	.0022	.0016
f2krigenh16ylinear	f2tricubic2	1.0790	.0210	.0154
f2krigenh16ylinear	f2trilinear2	2.0780	.0418	.0307
f2krigenh16znodrift	f2krigenh16zlinear	.1922	.0023	.0017
f2krigenh16zlinear	f2tricubic2	31.7500	1.2602	.9266
f2krigenh16zlinear	f2trilinear2	.0646	.0006	.0004
f2krigenh8nodrift	f2krigenh8linear	.9313	.0138	.0100
f2krigenh8linear	f2tricubic2	31.7500	1.2602	.9266
f2krigenh8linear	f2trilinear2	.0616	.0006	.0004

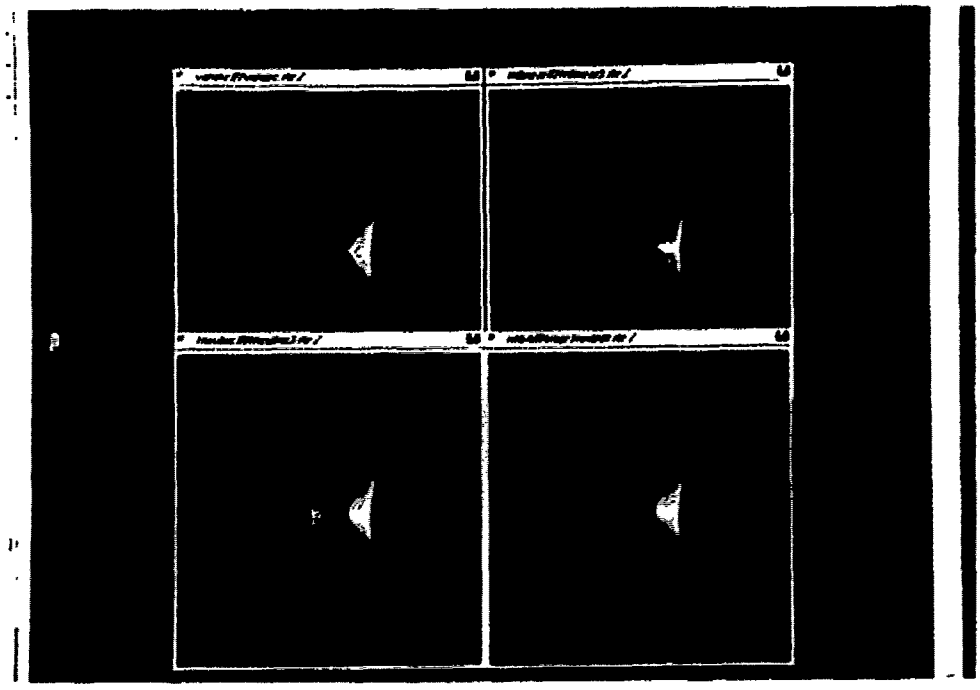


Figure 5.5. Subdivision factor 3. Upper left, vanilla marching cubes. Upper right, trilinear interpolation. Lower left, tricubic interpolation. Lower right, kriging with 64 neighborhood, no drift

local linear images match much closer to trilinear interpolation, rather than tricubic. For example, image f2krigenh16xlinear compared to image f2trilinear2 produces the smallest average difference percentage yet (.000457), for kriging images compared to the other two interpolation methods. This indicates that kriging with a neighborhood of 16 in directions 'X' or 'Y', local linear drift and the model semi-variogram h^2 estimates minor voxel values almost exactly as trilinear interpolation. The results derived from using kriging in a neighborhood of 8 are similar in quality to those derived by neighborhoods 16 'X' and 'Y'. This indicates that at least for this artificial volume data set, the additional values in the 16 'X' and 16 'Y' neighborhoods add virtually nothing of significance to the estimates.

Recall that I was attempting to tailor kriging to model tricubic interpolation. However, as the table and images indicate, without changing any parameters except neighborhood size, kriging also behaves like trilinear interpolation. It appears that kriging used with this particular data, is more influenced by neighborhood size and local drift assumptions rather than the semivariogram model. To test this further, I experimented with semivariogram models of h and h^2. I obtained average differences

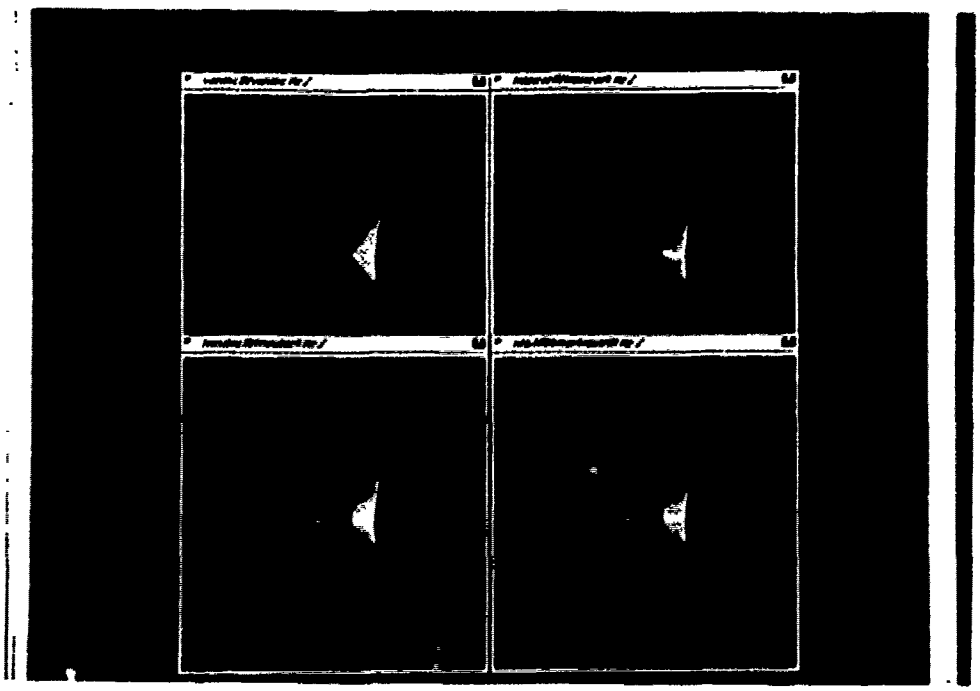


Figure 5.6. Subdivision factor 4. Upper left, vanilla marching cubes. Upper right, trilinear interpolation. Lower left, tricubic interpolation. Lower right, kriging with 64 neighborhood, no drift

of estimated values less than .0005%. The data was compared to data estimated by the h^3 model, holding all other assumptions the same.

The most accurate method is difficult to determine because the estimated values are so close and because later images using scanner generated medical data indicate that tricubic and the corresponding kriging method with tricubic like assumptions are not very accurate. For artificial volumes, such as the one containing this hyperboloid surface, the smoothest looking images are generated by the cell subdivision method using tricubic interpolation or kriging at higher subdivision factors.

The problem with considering only artificial volumes is that they do not adequately represent the world of medical images. In particular, there is no noise or sharp contrasts in voxel values. The next section explores how estimation techniques perform in medical image slice interpolation.

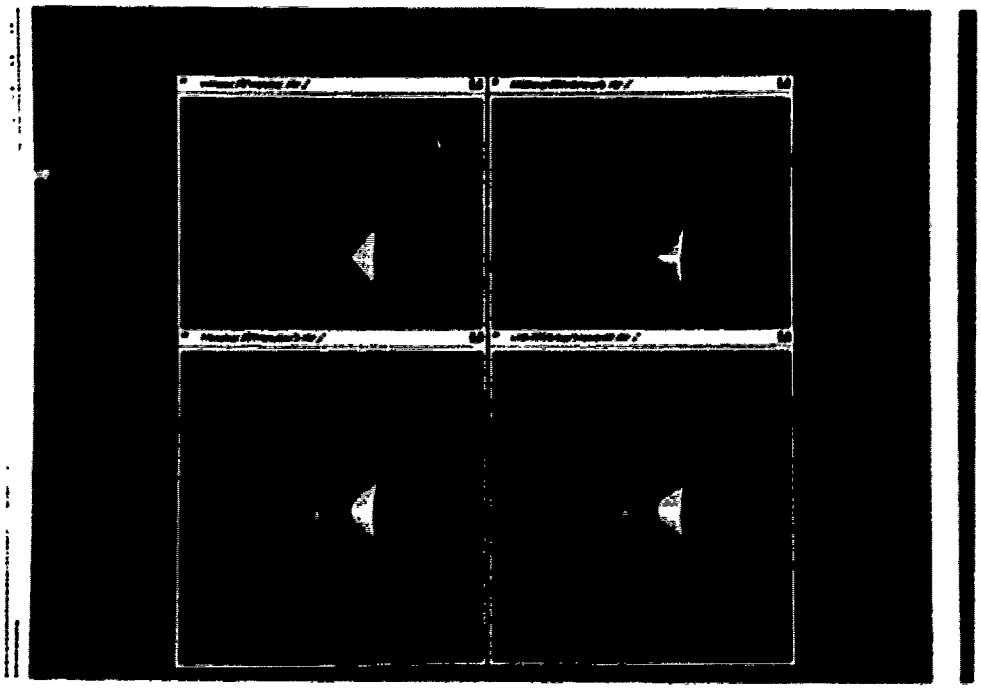


Figure 5.7. Subdivision factor 5. Upper left, vanilla marching cubes. Upper right, trilinear interpolation. Lower left, tricubic interpolation. Lower right, kriging with 64 neighborhood, no drift

5.2 Medical image slice interpolation

Medical image slice interpolation is a scene transformation[2]. In this section, 2D image data derived from MRI and CT studies is transformed from by the slice interpolation. Slice interpolation creates logical slices between existing ones by estimating scalar values in the 'Z' dimension. In this section, I demonstrate the use of linear interpolation, tricubic interpolation, and kriging as the interpolation methods used to estimate data values. Trilinear and linear are equivalent in this case because I interpolate in only one dimension, 'Z', along a cell edge.

5.2.1 Dog heart, CT. The first study consists of 202 X 132 pixel CT slices of a dog's heart, see picture in figure 5.14. The goal is to create a new slice between slices 41 and 42. The top four images in the picture are the original data slices. The remaining images are attempts to estimate a logical data slice between the two

[2]Recall that scene space is the data derived from medical imaging modalities in Herman and Udupa's terminology (see chapter two)

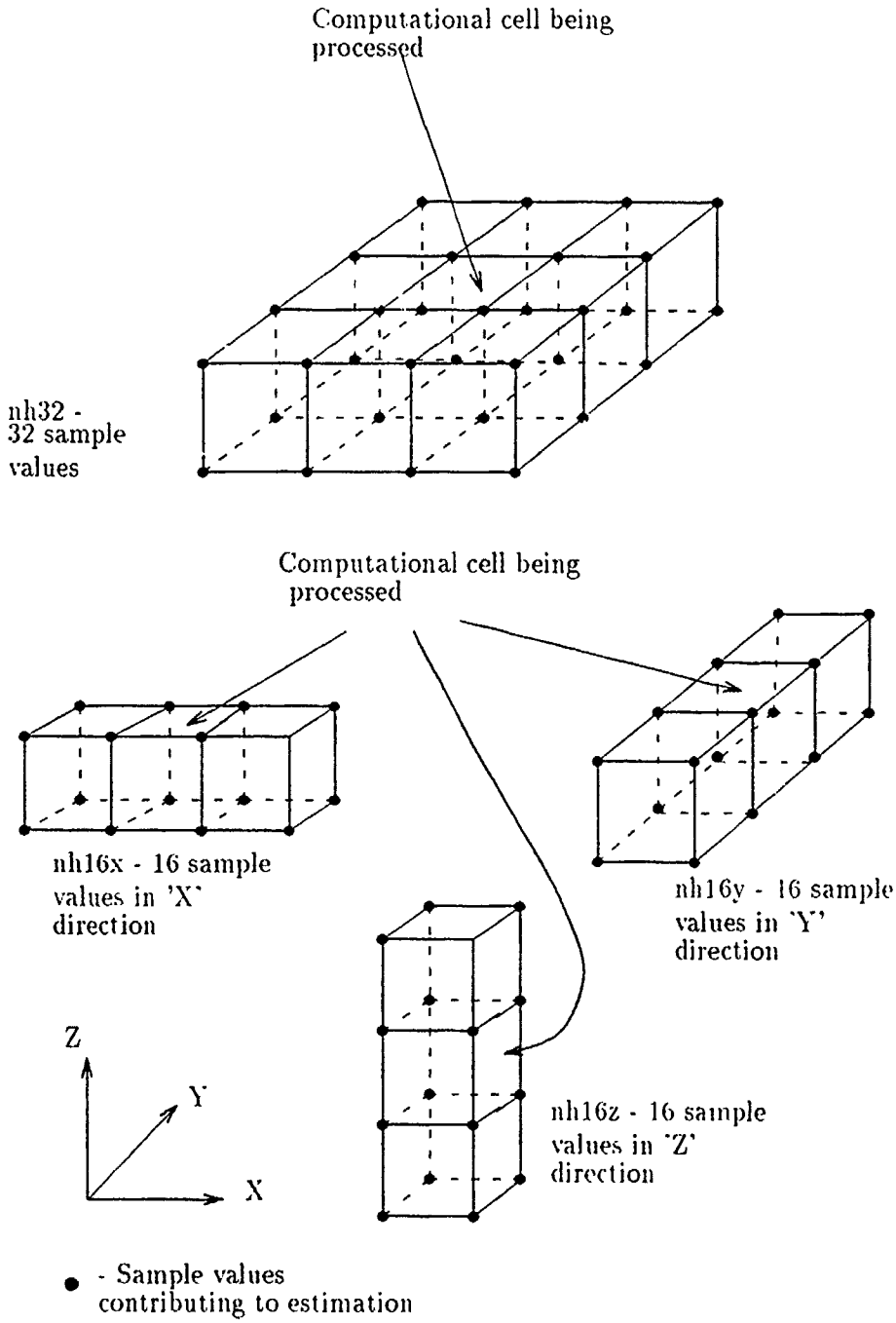


Figure 5.8. Kriging neighborhoods, nh32, nh16x, nh16y and nh16z

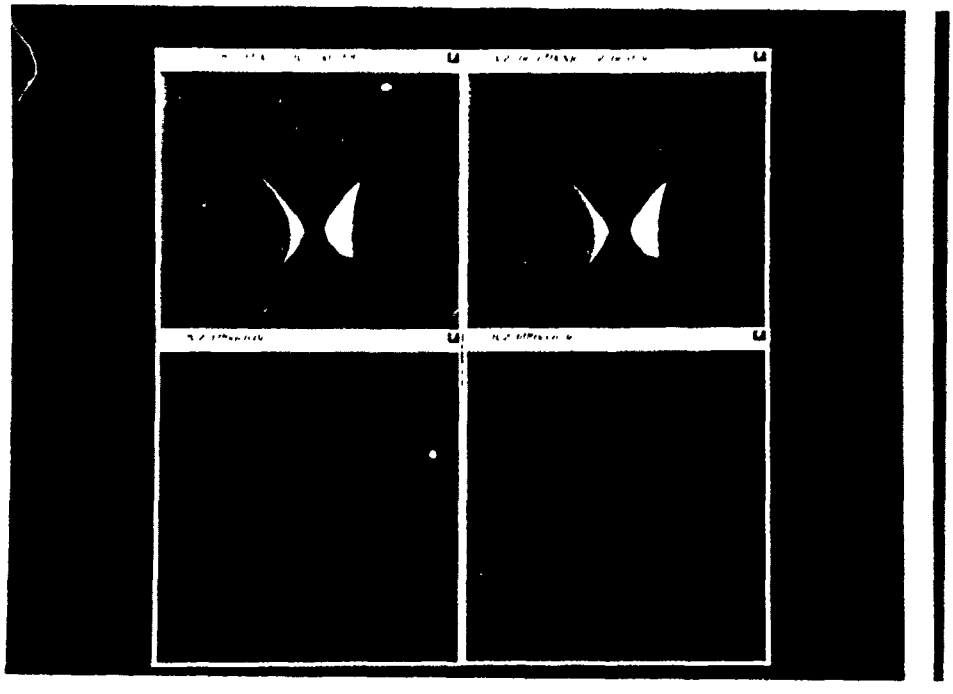


Figure 5.9. Kriging estimation. subdivision factor 2. neighborhood 32. Upper left, no drift. Upper right. local linear drift. Lower left. image difference of upper right from upper left. Lower right. image difference of upper right image from f2tricubic2.

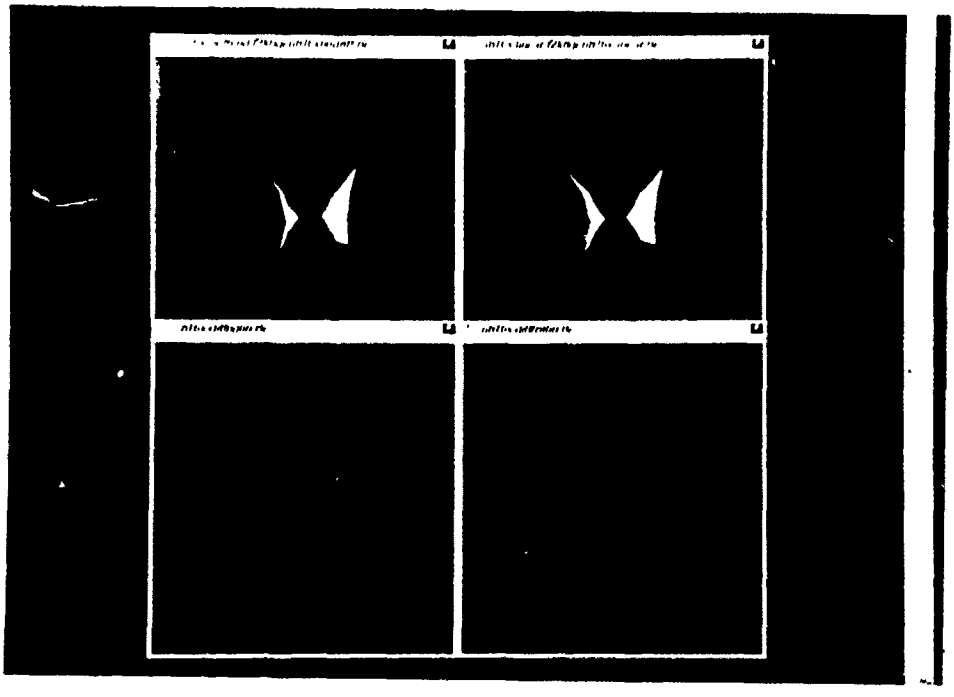


Figure 5.10. Kriging estimation, subdivision factor 2, neighborhood 16x. Upper left, no drift. Upper right, local linear drift. Lower left, image difference of upper right from upper left. Lower right, image difference of upper right image from f2trilinear2.

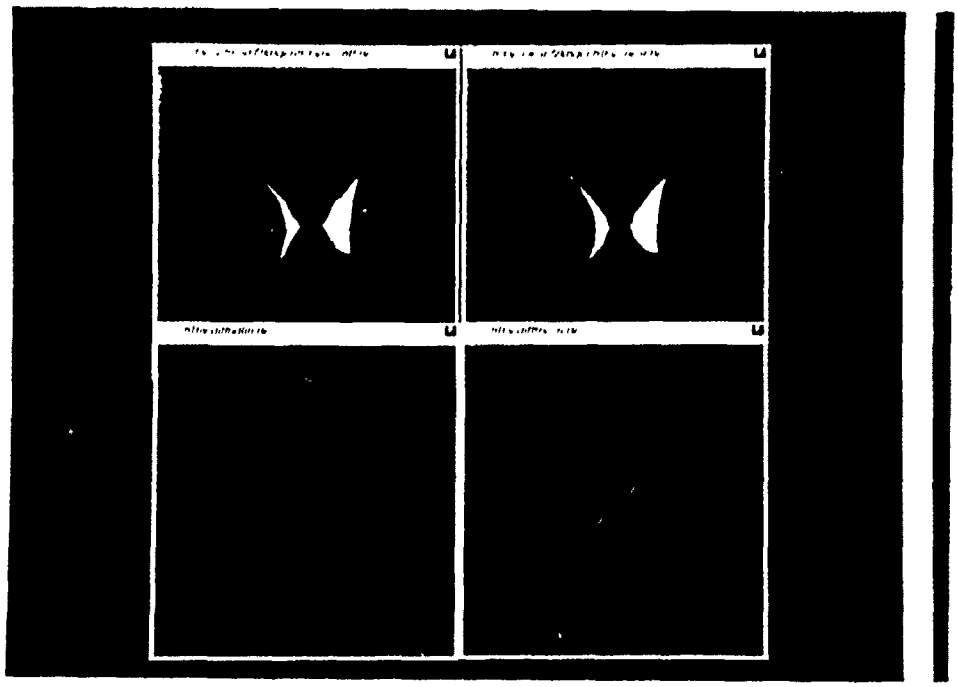


Figure 5.11. Kriging estimation, subdivision factor 2, neighborhood 16y. Upper left, no drift. Upper right, local linear drift. Lower left, image difference of upper right from upper left. Lower right, image difference of upper right image from f2tricubic2.

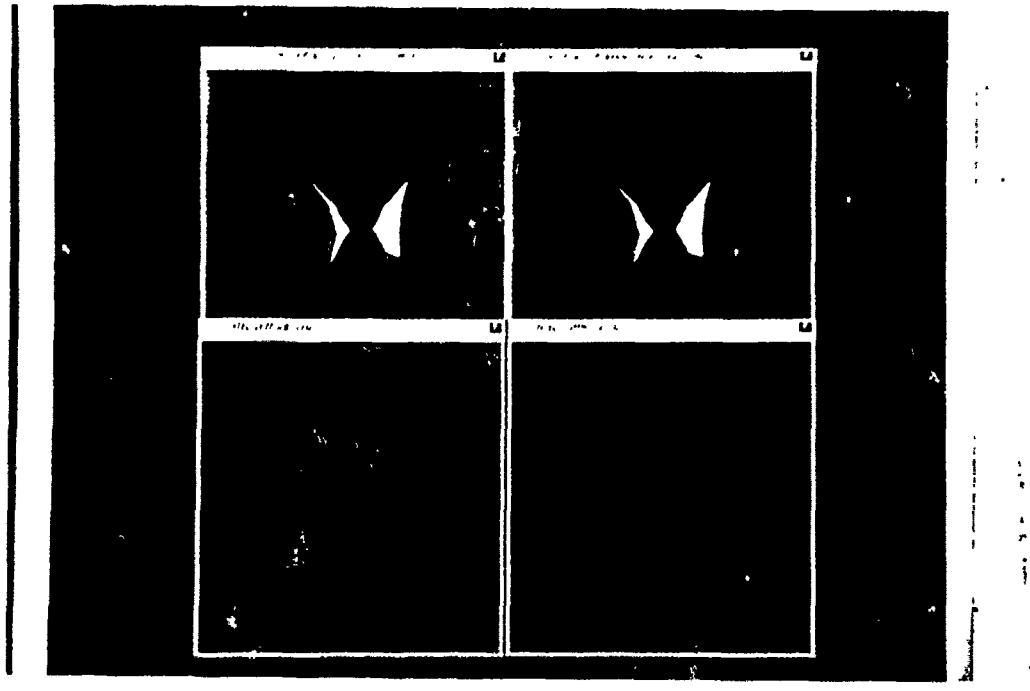


Figure 5.12. Kriging estimation. subdivision factor 2. neighborhood 16z. Upper left. no drift. Upper right. local linear drift. Lower left. image difference of upper right from upper left. Lower right. image difference of upper right image from f2trilinear2.

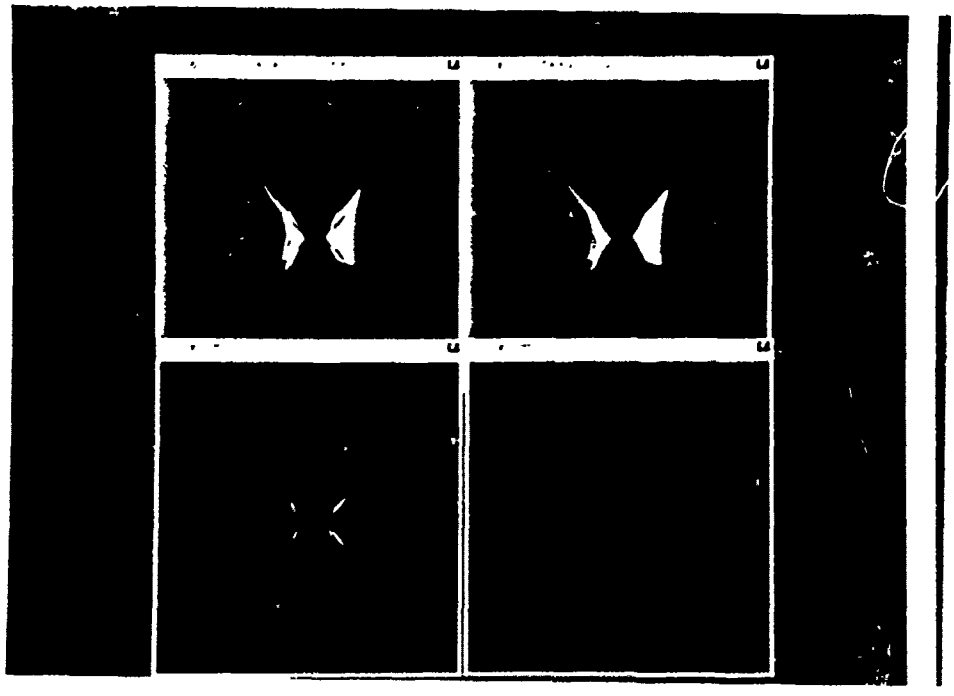


Figure 5.13. Kriging estimation, subdivision factor 2, neighborhood 8. Upper left, no drift. Upper right, local linear drift. Lower left, image difference of upper right from upper left. Lower right, image difference of upper right image from f2trilinear2.

middle images in the top row, using linear and tricubic interpolations and kriging (in various forms). Table 5.3 presents results pertaining to these images.

Notice in the picture the image generated by tricubic interpolation is inaccurate. The images produced by kriging with neighborhood 64 are also inaccurate and according to the comparison in the table, estimate values similar to tricubic interpolation. These inaccuracies are possibly caused by two reasons. First, the estimation functions might be accounting for values that should not be influencing the estimation. Second, the assumption of cubic variation might be invalid. Nothing can be changed with the tricubic interpolation function to prevent these inaccuracies. However, parameters in the kriging process can be modified to prevent them. These parameters are neighborhood size, drift assumptions, and the semivariogram. In this study, I only modify the drift assumptions and the neighborhood size.

Notice that except for kriging with neighborhood 16 'Z', kriging with smaller neighborhoods estimates values closer to trilinear than to tricubic, and gets the closest match to trilinear with a neighborhood of 8, assuming local linear drift. Ordinary kriging with smaller neighborhood sizes of 16 and 8 produces significant errors in the estimated values. However, universal kriging (local drift assumed) corrects these errors.

5.2.2 Baby head, MRI. The second 2D medical image study (see figure 5.15 consists of 166 X 166 MRI slices of a three month old baby's head. Interslice thickness is 4 mm. The goal is to estimate a new slice between slices 31 and 32. The original slices again are in the top row and the second row shows slices are derived by linear and tricubic interpolation. All others are different forms of kriging.

Although all the estimated images (except krigenh8nodrift) look almost identical, an examination of the estimated values reveals that estimations in this study are similar to those in the dog heart study (see table 5.4).

5.3 Medical volume cell interpolation surface extraction

The final series of images consist of a 3D surface extracted from 60 human baby head MRI data slices. The 2D slice dimensions are 128 pixels X 128 pixels and the iso-value is 43. The surface methods I use are the vanilla marching cubes (figure 5.16) and cell subdivision. For the cell subdivision technique, I use a subdivision factor of 2 in the 'Z' dimension and 1 in both the 'X' and 'Y'. This has the

Table 5.3. Comparison of dog heart CT estimated values.

Image	values compared with	largest est value diff	avg diff of values	avg diff / 256 %
krigenh64nodrift	krigenh64linear	.6229	.0196	.0077
krigenh64linear	tricubic	4.1242	.1365	.0533
krigenh64linear	linear	18.0780	.6781	.2649
krigenh32nodrift	krigenh32linear	.9815	.0416	.0163
krigenh32linear	tricubic	12.1747	.5519	.2156
krigenh32linear	linear	3.2616	.1295	.0506
krigenh16xnodrift	krigenh16xlinear	30.0089	1.3080	.5109
krigenh16xlinear	tricubic	12.4780	.5407	.2112
krigenh16xlinear	linear	2.2749	.0785	.0307
krigenh16ynodrift	krigenh16ylinear	70.8097	.9918	.3870
krigenh16ylinear	tricubic	12.7687	.5527	.2159
krigenh16ylinear	linear	3.0524	.1129	.0441
krigenh16znodrift	krigenh16zlinear	24.5295	.4502	.1759
krigenh16zlinear	tricubic	2.7525	.1125	.0439
krigenh16zlinear	linear	17.1736	.6742	.2634
krigenh8nodrift	krigenh8linear	203.1685	10.8014	4.2190
krigenh8linear	tricubic	14.4007	.5639	.2203
krigenh8linear	linear	.2923	.0091	.0036

Table 5.4. Comparision of baby head MRI estimated values

Image	values compared with	largest est. value diff	avg diff of values	avg diff ——— 256 %
krigenh64nodrift	krigenh64linear	1.8551	.0571	.0223
krigenh64linear	tricubic	6.7329	.3772	.1473
krigenh64linear	linear	23.0072	1.4960	.5844
krigenh32nodrift	krigenh32linear	3.4845	.0932	.0364
krigenh32linear	tricubic	17.1510	1.0481	.4094
krigenh32linear	linear	5.1335	.2807	.1097
krigenh16xnodrift	krigenh16xlinear	96.5233	3.4776	1.3580
krigenh16xlinear	tricubic	17.3192	1.0851	.4239
krigenh16xlinear	linear	4.2738	.2252	.0880
krigenh16ynodrift	krigenh16ylinear	70.1153	2.5202	.9845
krigenh16ylinear	tricubic	17.4163	1.0853	.4239
krigenh16ylinear	linear	5.2594	.2249	.0878
krigenh16znodrift	krigenh16zlinear	21.8722	1.0342	.4040
krigenh16zlinear	tricubic	4.1201	.2519	.0984
krigenh16zlinear	linear	21.4664	1.4106	.5510
krigenh8nodrift	krigenh8linear	227.1227	16.3677	6.3940
krigenh8linear	tricubic	17.9036	1.1631	.4543
krigenh8linear	linear	.7859	.0282	.0110

Figure 5.14. Dog heart CT medical image slice interpolation. Create new slice between slices 11 and 12. Window titles depict type of estimation performed.

effect of performing the volume pre-processing operation of slice interpolation. The estimation techniques employed are trilinear and tricubic interpolations and kriging. The kriging assumptions are local linear drift, a neighborhood of 8 and the h^3 model semivariogram. As can be seen by the pictures (figures 5.17, 5.18, and 5.19), the inaccurate estimates of tricubic interpolation in the volume pre-processing operation significantly affect the final reconstructed surface. Trilinear interpolation and kriging estimation techniques produce images without the obvious inaccuracies. The images produced by vanilla marching cubes, trilinear interpolation and kriging look very similar to each other. However, difference images between them indicate there are significant differences. For example, figure 5.20 shows the pixel differences between the images generated by using trilinear interpolation and kriging.

5.4 Chapter summary

In this chapter I presented the results obtained by implementing the three estimation techniques, tricubic interpolation, trilinear interpolation and kriging. First presented were images depicting a hyperboloid surface artificially embedded in a 4

Figure 5.15. Baby head MRI medical image slice interpolation. Create new slices between slices 32 and 33. Window styles depict type of estimation performed.

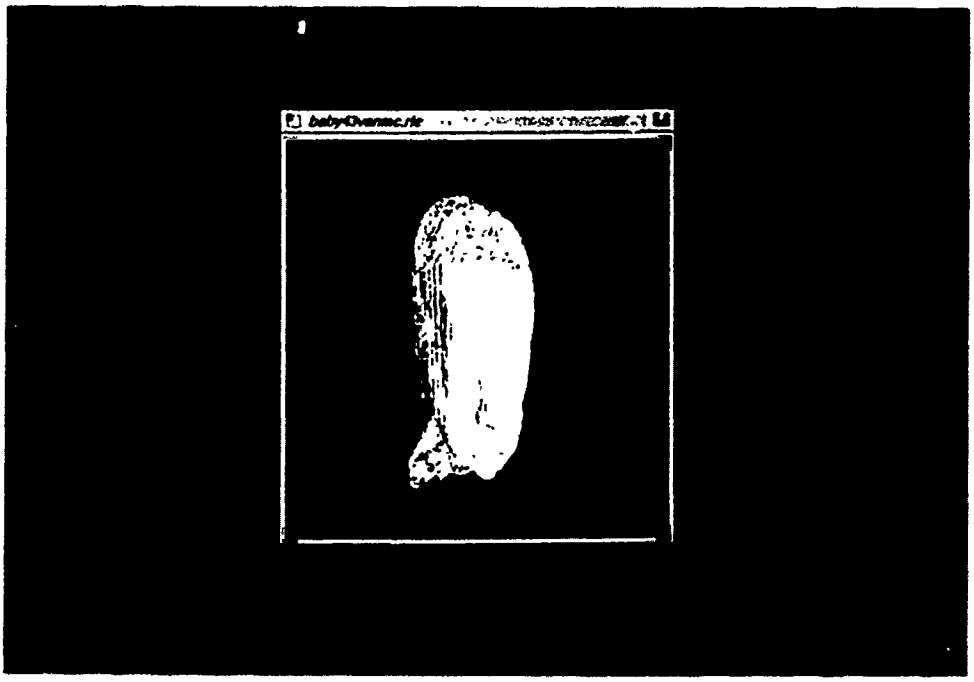
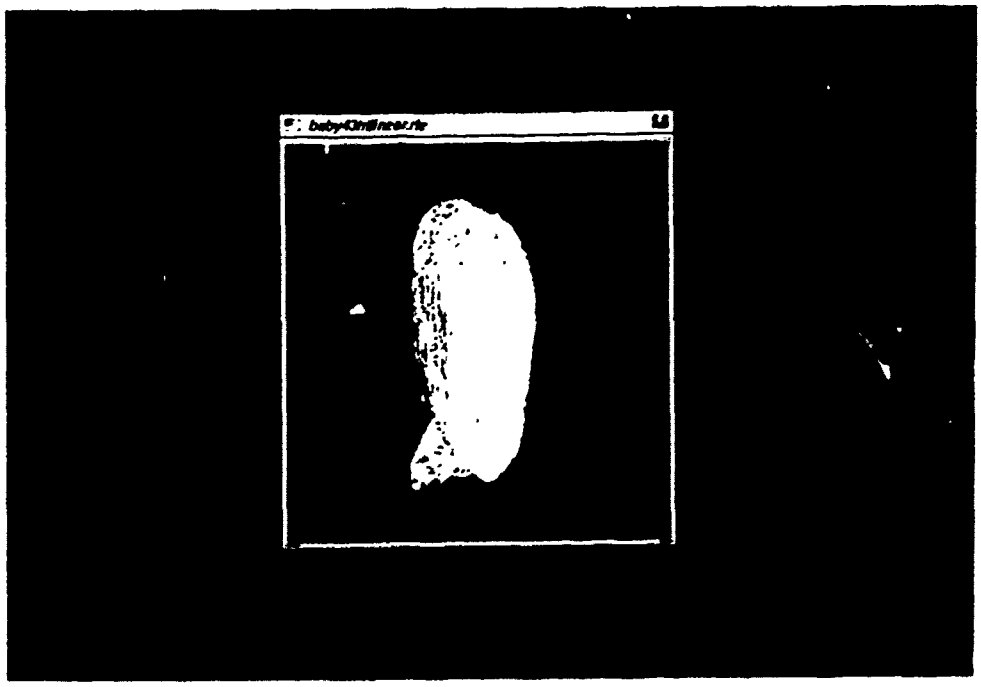
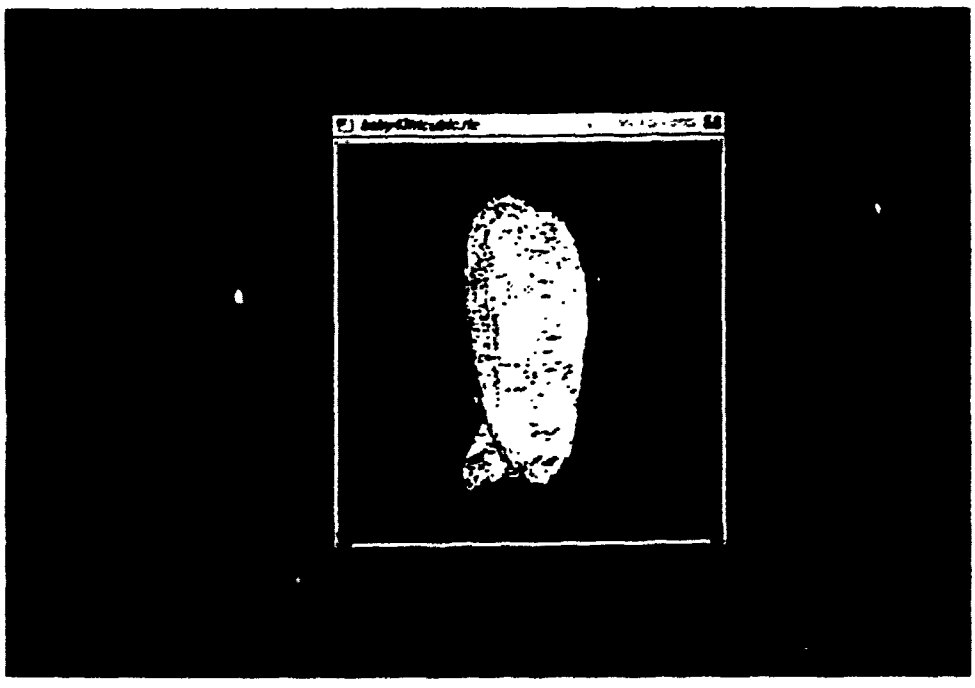


Figure 5.16. Baby skin iso-value 13, Vanilla Marching Cubes





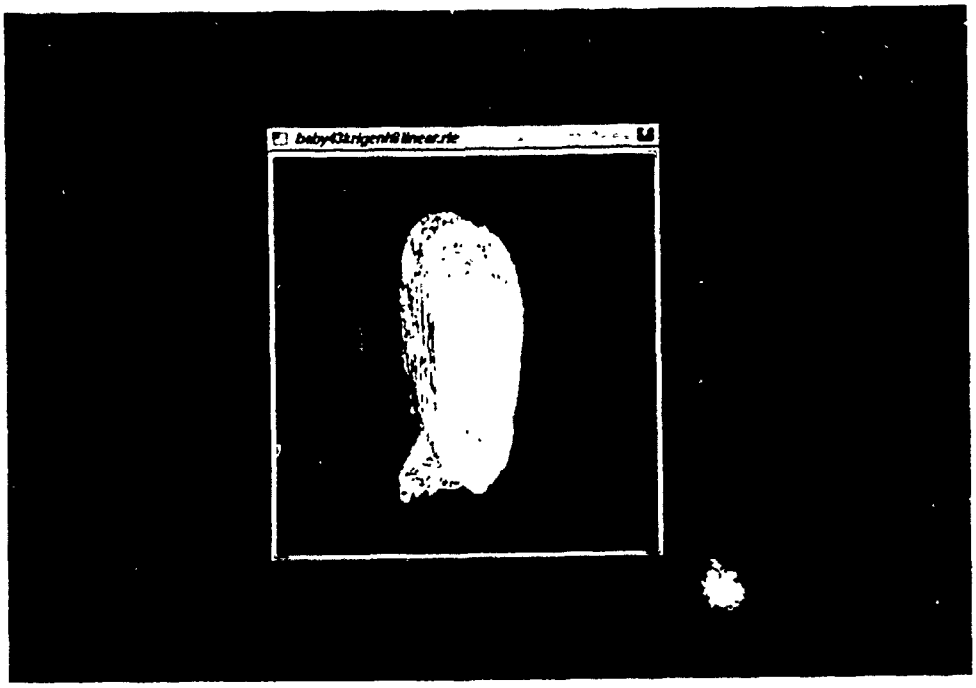


Figure 5.19 Baby skin,iso-value 13, Cell Subdivision, kriging

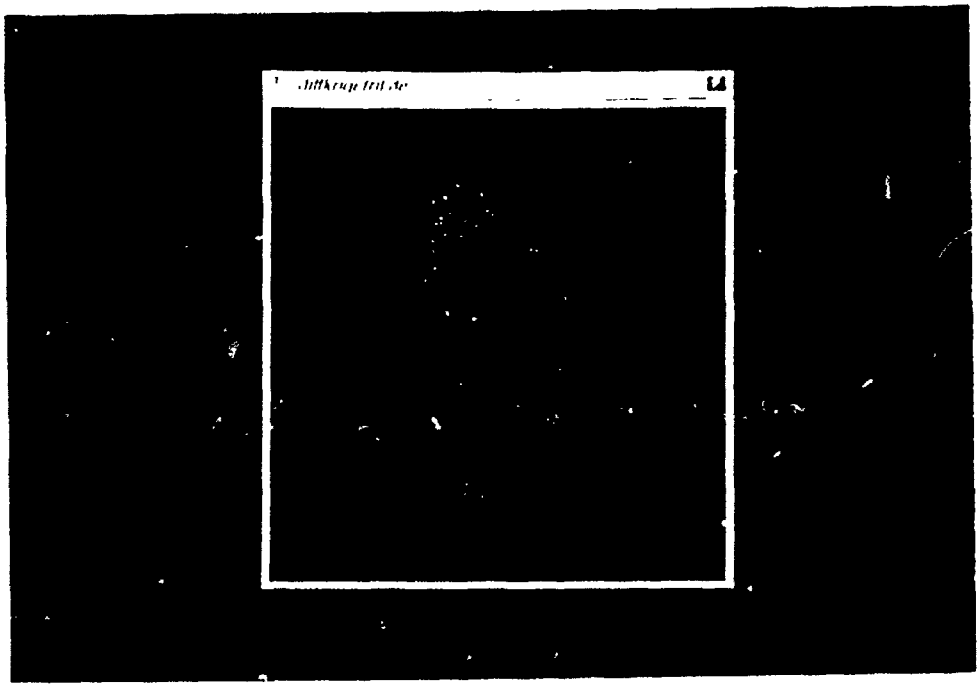


Figure 5.20 Baby skin,Difference image between images produced by trilinear interpolation and kriging

X by 4 Y by 4 Z volume. The ambiguous cell cases in the volume were removed in all cases by cell subdivision, regardless of the estimation technique employed or the subdivision factor. These images demonstrated that kriging is very flexible. That is, modifying the neighborhood size and assumed local drift significantly alters the way kriging estimates values. Larger neighborhood sizes of 64 and 32 cause kriging to behave like tricubic interpolation, regardless of the assumed local drift. However, at neighborhood sizes of 16 'X', 16 'Z', and 8, kriging with local linear drift behaves almost exactly as trilinear interpolation. The use of universal kriging was essential at these lower neighborhood sizes for correcting inaccuracies produced from using ordinary kriging. The best appearing images in the artificial volume occurred with tricubic interpolation and kriging tailored to behave like tricubic interpolation.

Different results were obtained by estimating 2D medical data slices between two existing ones. At neighborhoods of 16 'X', 16 'Y' and 8, kriging estimated more like linear interpolation than tricubic interpolation. Again, the use of universal kriging was critical for correcting gross inaccuracies produced by using ordinary kriging. The most significant result in these studies is that tricubic and kriging tailored to behave like tricubic, estimates values poorly. Little can be done to tricubic interpolation to fix these inaccurate estimations. It inherently assumes a cubic variation and uses 64 sample values for an estimate. However, kriging can be modified to overcome these inaccuracies. I did this by reducing the neighborhood size. The best image in the study was obtained with a neighborhood size of 8 and local linear drift. This image may not represent the best estimated values however. The best estimates can only be derived by doing a structural analysis of the data to determine if the data is isotropic, to find a possibly better model semivariogram, and to determine the optimal number of sample values. Furthermore, the optimal neighborhood size may not be 8, 16, 32 or 64.

Finally, I presented a study of images derived from 60 MRI slices reconstructed into a 3D surface representation. This study demonstrated that tricubic interpolation inaccurately estimates intra-cell scalar values for the purpose of cell subdivision surface extraction - at least for the data set I analyzed. Both trilinear and kriging, modified to behave more like trilinear than tricubic interpolation, produced much better appearing images.

Showing that kriging can estimate like standard deterministic methods is important. It shows that just guessing the kriging parameters produces results as well

as those already used in practice. This means that by using kriging, the estimation is no worse than the standard deterministic methods. However, kriging theory states that kriging will produce the best estimates if properly applied. Properly means performing a structural analysis of the data to determine a model semivariogram that models the spatial correlation of sample values.

Since kriging subsumes the other deterministic methods investigated, kriging alone could be used if different interpolation methods were desired for one rendition. This dynamic choice of an estimation technique would be most useful for interactive applications, where different estimation techniques might be useful at different resolutions. For example, consider the baby face skin rendition shown in this chapter. Trilinear interpolation or kriging tailored to behave like trilinear both appear to be good estimation techniques for this particular viewpoint and subdivision factor. However, suppose a radiologist is interested in viewing just the nasal area. Trilinear interpolation with this subdivision factor may not provide an accurate enough estimation for the detail required for a close up examination of this region. The subdivision factors could be adjusted as well as kriging parameters such as neighborhood size, semivariogram and local drift assumptions, to provide more accurate estimations for closer viewing.

VI. Recommendations and Conclusion

This chapter first discusses recommended research using the 3D imaging techniques implemented. Following that, future kriging research applied to 3D imaging is suggested. Finally, a brief conclusion to the thesis is presented.

6.1 3D Imaging Recommendations

The 3D imaging methods implemented in this work can be used in the following ways to aid future research. First, since the marching cubes implementation processes cells, it can be used as a foundation for any of the cell-by-cell processing techniques, e.g., (46), (50), and (32). Although the implementation currently maintains only four slices in memory at once, it can be modified to maintain all or any portion of the volume in memory simultaneously.

Another topic is different approaches to cell disambiguation. The simplistic method of facial averaging in the soft objects method (52) can be modified by applying different estimation techniques to estimate the center point of ambiguous faces. This was the focus of Wilhelm and Gelder's work (50). The software implemented in this research does not resolve ambiguous cells. Subdividing alone does not guarantee disambiguation like facial averaging does. The reason for this is after subdividing an ambiguous major cell, minor cells can be ambiguous. To obtain a smoother looking surface as well as resolving ambiguous cells, the best method would be to subdivide and apply a method like facial averaging to completely disambiguate minor cells. I suggest that any future research begin by implementing the facial averaging technique applied to major and minor ambiguous cells. Following that, the gradient consistency heuristics should be implemented for comparison. Finally, kriging should be investigated as a facial center value estimation method. Since my tessellate routine is shared between vanilla marching cubes and cell subdivision, estimating ambiguous center face values and disambiguating can be applied easily to both major and minor cells.

If a fast surface extractor is desired, several changes must be made to the implementation. The main bottleneck in the system is the I/O overhead incurred while writing and accessing geometry files on disk. Storing points and normals in a memory list and then passing them straight to a renderer should drastically cut the

processing time. Also, utilizing a hardware renderer should decrease the processing time even more.

The cell subdivision code could also be re-structured to operate in parallel. Since only four data slices are ever analyzed at a time, the parallelization could be partitioned by sets of data slices. Each set could be processed on a different processor to generate a portion of the iso-surface mesh. Rendering could also be partitioned among processors by the same logic.

6.2 Kriging Recommendations

The kriging estimation methods implemented in this work can be modified and enhanced to investigate further uses in 3D imaging. The results depicted and described in this thesis indicate scalar value estimation needs to be investigated further. Tricubic interpolation does not estimate values accurately in the medical data sets I analyzed. This study demonstrated that kriging is very flexible and can be modified to behave like different estimation methods, including both tricubic interpolation and trilinear interpolation. A research effort should be conducted to determine how to make kriging find the best estimation as the theory indicates it should. This requires a structural analysis of the data to determine the characteristics of the regionalized variables - support, continuity, and anisotropy. Continuity of sample values exist in certain portions of the human body such as organ and bone. What needs to be determined is how to find these zones of influence. Finding the zone of influence determines the kriging neighborhood and the type of model semivariogram to use. In addition to the model semivariogram, I also made assumptions about the drift, the neighborhood, and isotropy. In some cases, such as a neighborhood of 64, the data might be anisotropic. If assumption of isotropy or the model semivariogram was wrong, fixing them according to a structural analysis should make kriging produce "optimal" estimates.

Before structural analysis can be done however, certain tools have to be built or modified from existing ones. Several structural analysis tools have been built at AFIT for use in 2D data sets. These would have to be modified for 3D use. The tools include one that calculates the experimental variogram. It currently estimates the semivariogram in only two directions (to check for anisotropy). This would have to be modified to estimate semivariograms in other directions for 3D. A procedure exists that determines parameters for a semivariogram model. There are only a few

models available in the tool, so model implementation is another area of research as well as modifying the existing ones for use in volume data sets.

6.3 Conclusion

This research investigated different methods for estimating scalar values within computational cells and in the volume pre-processing operation of slice interpolation. These methods include the deterministic linear, trilinear and tricubic interpolations and the geostatistical estimation technique, kriging. Iso-surfaces were generated by marching cubes and another cell interpolation method called cell subdivision. The estimation techniques were used to estimate intra-cell scalar values in the cell subdivision method. They were also used to estimate logical data slices between existing ones for the volume pre-processing operation of slice interpolation. This research introduced kriging as an estimation technique for use in 3D imaging.

I demonstrated that kriging estimates values as accurately as deterministic tricubic interpolation – shown to be very accurate in estimating intra-cell scalar values in artificial volumes. I also showed that tricubic interpolation can perform poorly in medical data sets and that kriging can be modified so these inaccuracies do not occur.

The erroneous results produced by tricubic and several of the kriging variations could be caused by invalid assumptions about the neighborhood influencing the estimation and the variation between sample values. Neither of these factors can be modified in tricubic interpolation; however, they can be in kriging. I only modified the neighborhood size and local drift assumptions. These modifications demonstrate that kriging produces better results than tricubic interpolation. The data variation, which is modelled by both the semivariogram and the local drift, needs to be determined by a structural analysis of the data. My goal was to demonstrate that kriging is capable of being modified to behave like other deterministic interpolation techniques and to prevent inaccuracies. Since I did not do a structural analysis and just assumed the data variation, the results show that kriging is very robust. It is robust because even though I guessed several of the kriging parameters, I was able to make kriging behave like three other standard deterministic functions and better in some cases. Following a structural analysis, kriging should provide the best estimation in comparison to other known estimation methods.

The ability to modify kriging to behave like any other deterministic method is important. First it shows that kriging can be modified to behave like standard estimation techniques, so if they are desired, kriging does no worse. That is, kriging subsumes the deterministic methods I investigated. Also, kriging provides the capability to dynamically change the estimation technique. This capability could be used interactively to adaptively refine the estimation for different resolutions of data and/or for different viewpoints of the rendition.

Kriging is considered the optimal estimator and since accuracy is important in 3D medical imaging, exploring the use of kriging to estimate values is worthwhile. Its use has mainly been to estimate values within environments such as mining, gas and oil exploration and other geo-science disciplines. Further research is critical to prove the usefulness of kriging in 3D medical imaging. This research is also applicable to any 3D imaging methods that perform estimation.

Appendix A. *Vanilla Marching Cubes Data Flow Diagrams and Program Description*

A.1 Data Flow Diagrams

A.2 Program Description Language Statements

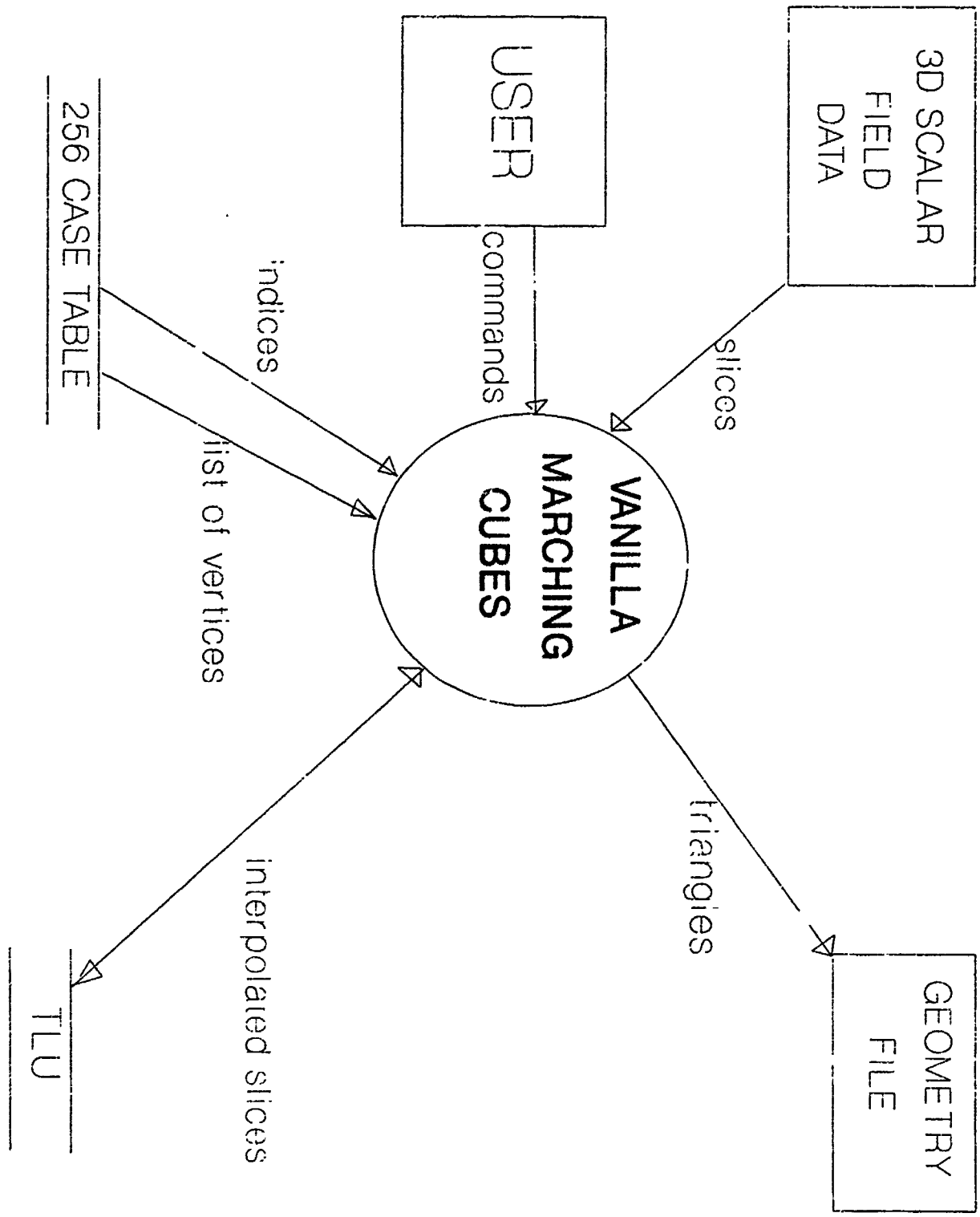


Figure A.1. Vanilla Marching Cubes Context Diagram

Figure A.2. Vanilla Marching Cubes Top Level DFD

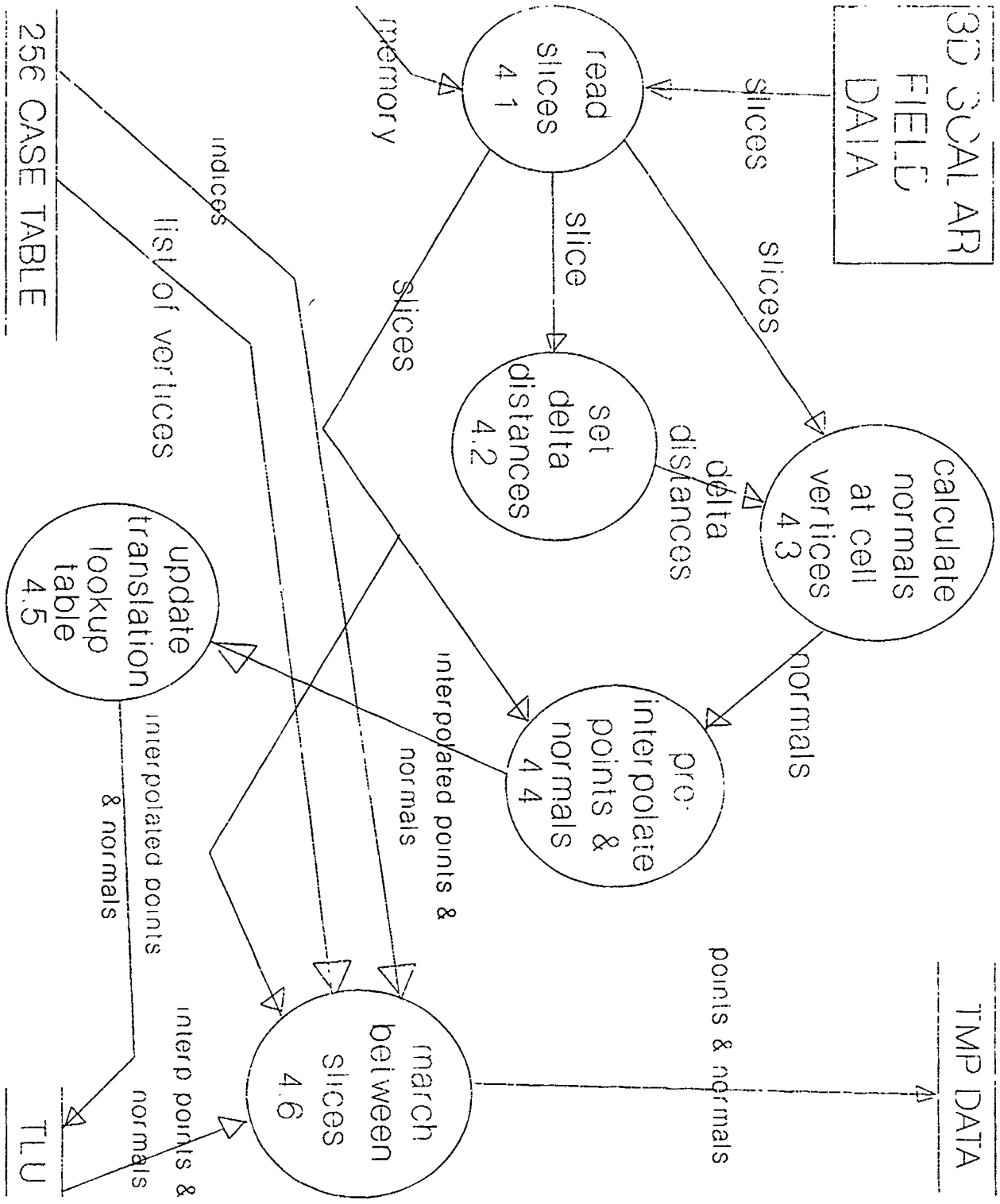


Figure A.3. Vanilla Marching Cubes "process slices" DFD

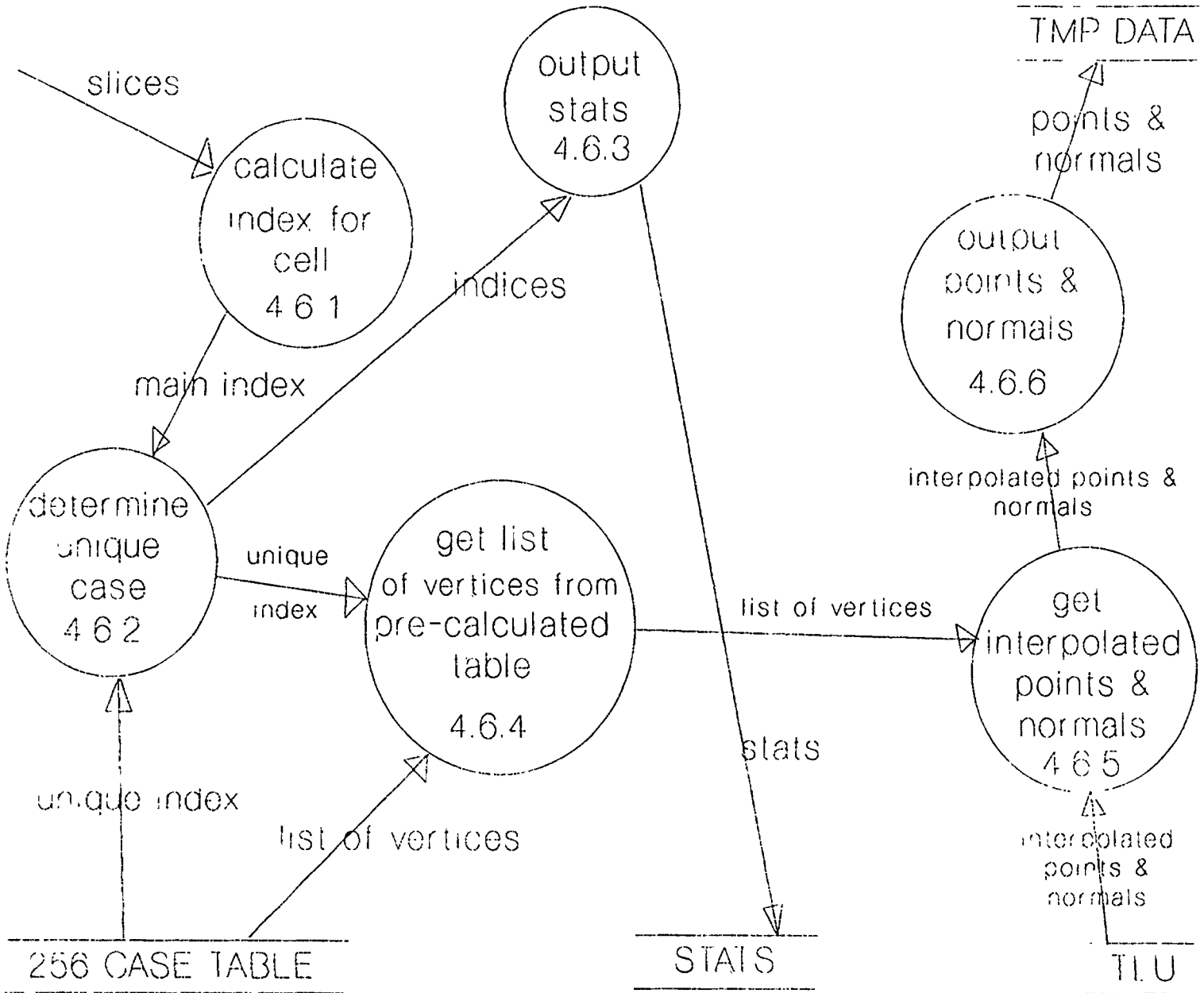


Figure A.4. Vanilla Marching Cubes "march between slices" DFD

Primary Data Structures :

- *plane[4] - Pointers to four arrays holding original points and scalar values. Need four in memory to calculate normals using gradient operator.
- *ixPlane[2] - Pointers to two arrays holding pre-interpolated 'x' points for cells between two slices.
- *iyPlane[2] - Pointers to two arrays holding pre-interpolated 'y' points for cells between two slices.
- *izPlane - Pointer to an array holding the pre-interpolated 'z' points for cells between two slices.
- *Gplane[2] - Pointers to two arrays holding calculated normals at cell vertices.
- *gxPlane[2] - Pointers to two arrays holding pre-interpolated 'x' normals for cells between two slices.
- *gyPlane[2] - Pointers to two arrays holding pre-interpolated 'y' normals for cells between two slices.
- *gzPlane - Pointer to an array holding pre-interpolated 'z' normals for cells between two slices.
- 256 case table.
- interpolated points translation table. Indexed by cell edge vertices.
- interpolated normals translation table. Indexed by cell edge vertices.
- isovalue array.

A.2.1
parse command line ("Top Level" DFD, newline Bubble 1)

```
while no more command line arguments supplied by user
    switch (command) {
        case : isovalue_flag {
            read isovalue argument:
        }
        case : shell_flag {
            read file name argument:
            read isovalues from file. Enter isovalues into an
            array of isovalues:
        }
```

```
            case : window_flag {
                read range of isovalues:
                set as first and second elements in array of isovalues;
            }
            case : box_flag {
                set box flag to true:
            }
            case : list_of_geom_files_flag {
                set list of geom files flag to true:
            }
            case : path_flag {
                read path argument:
                set path variable to argument, overriding default path:
            }
            case : data_file_flag {
                read from control file to get large data file information
                such as image dimensions, interslice amount,input data path, and
number of slices.
                /*else default is to read from an artificial volume data file:*/
            }
        } /* switch */
    } /* while */
```

A.2.2

read header info ("Top Level" DFD, Bubble 2)

```
    check data_file_flag and open appropriate file:
    retrieve x and y:
    if data_file_flag indicates an artificial volume, retrieve z dimension:
    else if large date files, retrieve number of slices from control file.
    /* Number of slices becomes the Z dimension */
```

A.2.3

allocate memory storage ("Top Level" DFD, Bubble 3)

Allocate memory based on dimensions;

/* Example memory usage for arrays holding original and

intermediate slice data.

plane arrays consist of structures with 4 floats.

all other arrays besides the isovalues array has 3 floats per structure.

Assume 256X256 medical images as input.

4 plane arrays x image size x 4 floats = 4 x 256 x 256 x 16 = 4MB

12 normal and interpolated arrays x image size x 3 floats =

12 x 256 x 256 x 12 = 9 MB */

A.2.4

process slices ("Top Level" DFD, Bubble 4)

A.2.4.1

read slices ("process slices" DFD, Bubble 4.1)

Initially read in 3 slices into plane[0].plane[1],plane[2]:

/* Done to process border case, marching between plane[0] and plane[1]. */

Then in main for loop, read fourth slice, march ;

A.2.4.2

set delta distances ("process slices" DFD, Bubble 4.2)

/* done after reading in first three slices, before main for loop */

$\Delta X = abs(plane[0]->x - (plane[0]+1)->x)$;

$\Delta Y = abs(plane[0]->y - (plane[0]+xdimension)->y)$;

$\Delta Z = abs(plane[0]->z - plane[1]->z)$;

/* note, this assumes the delta distances will not change between slices

and within each slice, i.e., assumes a regular data grid. */

A.2.4.3

calculate normals at cell vertices ("process slices" DFD, Bubble 4.3)

Initially calculate for cells between plane[0] and plane[1]:

In main for loop calculate for cells between plane[1] and plane[2]:

In final case, calculate for cells between plane[2] and plane[3]:

A.2.4.4

pre-interpolate points & normals ("process slices" DFD, Bubble 4.4)

Initially calculate for cells between plane[0] and plane[1]:

In main for loop calculate for cells between plane[1] and plane[2]:

In final case, calculate for cells between plane[2] and plane[3]:

Method : Use same comparison as in marching cubes for loop -

If a points' scalar value is greater than the isovalue, it's a 1-voxel;

if one voxel is classified as a 1-voxel and another adjacent

voxel of the cube is a 0-voxel, then interpolate along the

edge and store in the appropriate array (which array depends on edge);

A.2.4.5

update translation lookup table ("process slices" DFD, Bubble 4.5)

Prior to marching cubes (cells) between two data slices, re-establish

pointers in translation lookup tables;

/* These pointers must be re-established because the original data

slice pointers and the normal slice pointers are swapped after every

new slice is read in from the data file. The translation tables are

indexed by the cell edges and the entries contain pointers to the first

cell's edges. When the tables are used, an offset from the beginning of

the arrays is added to the pointers to access the correct cell.

For example $itlu[5][7] = izPlane + xdimension + 1$ establishes the

absolute cell address for the interpolated point of cell edge 5-7.

A.2.5

march between slices ("process slices" DFD. Bubble 4.6)

```
/* input are pointers to two arrays of consecutive slices (planes). */
/* main marching cubes loop */
for(iy = 0; iy < ydim; iy++)
    for(ix = 0; ix < xdim; ix++) {
        establish offset from plane pointers to lower left vertex of
        current cell;
        establish all other cell vertices from lower left vertex;
        calculate index for cell ("march DFD", Bubble 4.6.1);
        determine unique case ("march DFD", Bubble 4.6.2);
        switch (unique index) {
            for each of the 15 unique cases do :
                output stats ("march DFD", Bubble 4.6.3);
                get list of vertices from pre-calculated
                table ("march DFD. Bubble 4.6.4)
                get interpolated points & normals ("march DFD",
                Bubble 4.6.5)
                output points & normals ("march DFD",
                Bubble 4.6.6)
        } /* end switch */
    } /* end for ix */
```

A.2.5.1

calculate index for cell ("march" DFD. Bubble 4.6.1)

```
test each cell vertex to see if it's scalar value is greater than the
isovalue(s);
If a vertex is so classified, boolean OR a flag with that vertex #:
Result of flag after all vertices are classified is the index into the
256 case table.
```

A.2.5.2

determine unique case ("march" DFD, Bubble 4.6.2)

Access 256 case table with cell index, retrieve unique case #;

A.2.5.3

output stats ("march DFD", Bubble 4.6.3)

Tally unique case #, ambiguous case #, # of triangles and # of points;

A.2.5.4

get list of vertices from pre-calculated table ("march" DFD, Bubble 4.6.4)

Access 256 case table with cell index to retrieve list of vertices;

Set a temporary pointer to vertex list array;

A.2.5.5

get interpolated points & normals ("march" DFD, Bubble 4.6.5)

/* For each unique case, the cell edges for interpolation were identified and appropriate triangulations were selected. For example, unique case 3 requires interpolations along edges 0-4, 0-2, 1-3, and 1-5. Two possible triangulations can be chosen, but would not alter the image, so the choice is arbitrary in this case. Now that the cell edges to interpolate along are known, the translation lookup tables can be accessed with the offset calculated above. */

Access translation lookup table to retrieve pre-interpolated points and normals along cell edges.

A.2.5.6

output points & normals ("march" DFD, Bubble 4.6.6)

Output points and normals to a temporary points file;

Output line numbers specifying points for triangles. Place in temporary triangles file;

/* temporary files used because AFIT geometry files requires all the points be listed first, followed by the point line numbers for triangles.

A.2.6

write box ("Top Level" DFD, Bubble 5)

/* only applicable to artificial volumes and only visible when viewing triangle mesh */

If box_flag entered by user, write rectangles forming a box surrounding triangle mesh to temporary files:

A.2.7

output geometry file ("Top Level" DFD, Bubble 6)

Catenate the two temporary files, with the appropriate header and trailer information attached :

A.3 Data Dictionary For Data Flow Diagrams

Data Dictionary Symbology:

=	is composed of
+	and
( )	optional
{ }	iteration, {}+ indicates 1 or more, default, zero or more.
* *	comment

256 CASE TABLE = indices + {list of vertices}+
* the pre-calculated table of cell vertices for the 256 different cell cases. Each of the 256 cases contains a mapping to a unique index and a list of vertices corresponding to the unique index. *

3D SCALAR FIELD DATA = * artifical volumes, medical image files, or scientific volume data. *

box = * lines drawn around dimensions of image generated from artificial volume. *

commands = (isovalue_flag) + (shell_flag) + (window_flag) + (box_flag) + (list_of_geom_files_flag) + (path_flag) + (data_file_flag)
* user must supply one of isovalue_flag, shell_flag or window_flag
isovalue_flag - argument is a single isovalue.

shell_flag - argument is a file name. File contains list of iso-values.
window_flag - argument is a range of isovalues. e.g., -w 30.0-40.0
box_flag indicates a box will be drawn around image – only used for
artificial volumes such as embedded math functions. *
list_of_geom_files_flag specifies that multiple geom files will
be generated. *
path_flag - specifies path for output geometry files. Default is the
directory path of the source code. *
data_file_flag - specifies type of input file(s) - 3D SCALAR FIELD DATA.
Can be medical or artificial volumes. Artificial volume data is default
and requires re-directing stdin using ¡. For example

delta_distances = * The lengths of cell edges. *

GEOMETRY FILE = AFIT geometry file of points triangle specifications.

header_info = data_dimensions + number_of_slices + (inter_slice_thickness) + (intra_slice_thickness)

indices = main_indices + unique_indices * indices into pre-calculated 256 CASE TABLE. *

interpolated points & normals = * points and normals interpolated along cell edges from cell vertices to the isovalue(s) provided by the user. *

list of vertices = * A list of vertices is pre-calculated manually for each case in the 256 CASE TABLE. and stored in the table. Each list contains the eight cell vertices. ordered according to unique case ordering (following the appropriate complementation and/or rotation(s)). *

points and normals = * points - The original points obtained from the 3D SCALAR FIELD DATA. normals - Normals approximated at cell vertices using central difference gradient operators. *

slices = points + scalar_values

stats = tally_ambiguous_cases + tally_main_cases + tally_unique_cases + tally_number_points + tally_number_triangles

triangles = points and normals

TLU = Translation Lookup Table
* Contains pointers to interpolated points and normals.
* see update translation table lookup PDL for more info. *

TMP DATA = tmp_points + tmp_normals + tmp_triangle_specs

Appendix B. *Vanilla Marching Cubes*

This appendix describes the implementation of the vanilla marching cubes algorithm developed by Lorensen and Cline in 1987 (34). It is termed vanilla because my implementation does not include any enhancements such as texture mapping, geometric solid modelling, or disambiguation. The first part of this appendix introduces the topic. Following the introduction, some background information about decisions and about my implementation are discussed. Then, the main steps in the implementation are listed. Next some background information on my implementation is discussed. Following this, use of the 256 element transformation table is explained. After this, the cell edge interpolation method is presented. Then the method to determine triangle normals is discussed. Then, modifications and enhancements made to the public domain marching cubes code is presented. Lastly, comments on the format of the marching cubes output are discussed.

B.1 Introduction

The marching cubes algorithm (34) is a 3D imaging method that extracts a surface of interest from a 3D volume of data. The surface is represented as a 3D triangular mesh and rendered by a standard polygonal based graphics renderer. The algorithm processes cubes or computational cells, where a cell is composed of eight voxels, four each from two adjacent data slices. Each cell is analyzed to determine if the surface of interest intersects the cell. Triangles are generated within cells

that are found to contain a portion of the surface. Surface detection within a cell is performed by simple thresholding. If a cell vertex is greater than the threshold value (iso-value), it is assigned a one (considered a 1-vertex), else it is assigned a zero (0-vertex). Vertex classification in this manner yields 256 possibly different cell classifcations. Lorensen and Cline reduced this to 15 unique cell cases. Triangle vertices are determined by linearly interpolating the voxel 3D points to the isovalue between 1- and 0-vertices. Normals are similarly calculated by interpolating previously derived cell vertex normals.

B.2 Background

In this research, I implemented a version of the marching cubes algorithm before deciding on a thesis topic. I then explored the use of kriging to somehow improve the surface extraction. I read Wilhelms and Gelder's (50) work on intra-cell scalar value estimation. From there it was simple to see how kriging could be used. But, I still investigated other surface extractors to determine the history of marching cubes and to see whether kriging could be applied in other areas. This led me into the area of 3D medical imaging, from where marching cubes as well as many other 3D imaging algorithms were derived.

As seen from chapter two, the two primary surface methods are cuberille based and cell interpolation. The cuberille-based approaches were developed by Herman, Liu and Udupa in the late 1970's and early 1980's. In contrast to the cuberille-based

methods. the cell interpolation methods are more heuristically based. Two of the cell interpolation algorithms are Lorensen and Clines' marching cubes (34) and (8) and Wyvill and McPheeters soft object algorithms (52). The latter two methods are called cell interpolation techniques because they interpolate polygonal vertices to the isosurface boundary along cell edges, where a cell is a parallelepiped with eight adjacent voxels as vertices. Cells are also known as computational cells. The important distinction between voxel-based surface extractors such as cuberille based models and cell-based surface extractors is the former assume a constant scalar value throughout the volume element (the voxel), whereas the latter assume a varying scalar value throughout cells.

After reviewing the 3D imaging literature, I chose to continue exploring the cell interpolation methods for the following reasons. First, I did not wish to develop a specialized model which would limit its use or future research to potentially only one application. Cell interpolation methods have been widely used in both medical imaging and scientific visualization. Additionally, the graph-theoretical methods used in the cuberille model are very complex to implement and often yield results that are jagged in appearance (because of the 2D surface display unit used - a cube face) without special shading procedures used (1). In contrast, the cell interpolation methods are much simpler to understand and implement and can yield very high quality images without special shading methods used, beyond the traditional approaches such as Gourard or Phong(13). Most importantly, since estimation is a

critical part of 3d visualization, the ability to estimate varying scalar values within a cell and visualize direct results of the estimation process provides the impetus to investigate improved estimation methods. The work previously done by Wilhelms and Gelder (50) provides a strong framework within which to visualize new estimation methods. The work done here should have potential benefit to many other 3d visualization methods, since processing cell-by-cell is common to other methods such as direct volume rendering (46) and (51).

The main drawback of the cell interpolation approaches is the simplistic method of segmenting the object of interest from the remainder of the volume. This method, termed thresholding, makes a binary decision at each voxel - does the voxel contribute to the final image or not? Volume methods do not make such a simple decision, but rather, allow voxels to contribute percentages of different characteristics (such as color, and density) to the final image (33) and (16). Again, the emphasis of this research is not to compare volume and surface methods, but to compare estimation techniques applicable to both methods.

I began my work with a public domain version of a basic marching cubes algorithm. This version only set up the precalculated table and performed the interpolation step, but did not do the most difficult task, that of computing normals from gradient information. It basically did steps 2-5 in the list presented in the next section, but as noted later, I re-wrote most of the code to make it more understandable and modifiable. As stated in chapter 2, the marching cube algorithm "marches"

computational cells between two slices of data. See figure B.1 for a pictorial representation of the marching. The slices are labelled according to the order of arrays I maintain in the implementation.

B.3 The Main Steps in the Implementation

Lorenson and Cline (34) described the marching cubes algorithm in 1987. The significance of their algorithm was that it used 3D information to construct interslice polygons to represent the iso-surface and to approximate normals for shading. The following is a list taken from the 1987 article, which denotes the steps performed in the marching cubes algorithm:

1. Read four slices into memory.
2. Scan two slices and create a cube from four neighbors on one slice and four neighbors on the next slice.
3. Calculate an index for the cube by comparing the eight density values at the cube vertices with the surface constant.
4. Using the index, look up the list of edges from a precalculated table.
5. Using the densities at each edge vertex, find the surface-edge intersection via linear interpolation.
6. Calculate a unit normal at each cube vertex using central differences. Interpolate the normal to each triangle vertex.
7. Output the triangle vertices and vertex normals.

The remainder of this section discusses how some of the above steps were implemented. I do not discuss the steps that are straightforward from an understanding of the basic algorithm described in chapter two of the thesis.

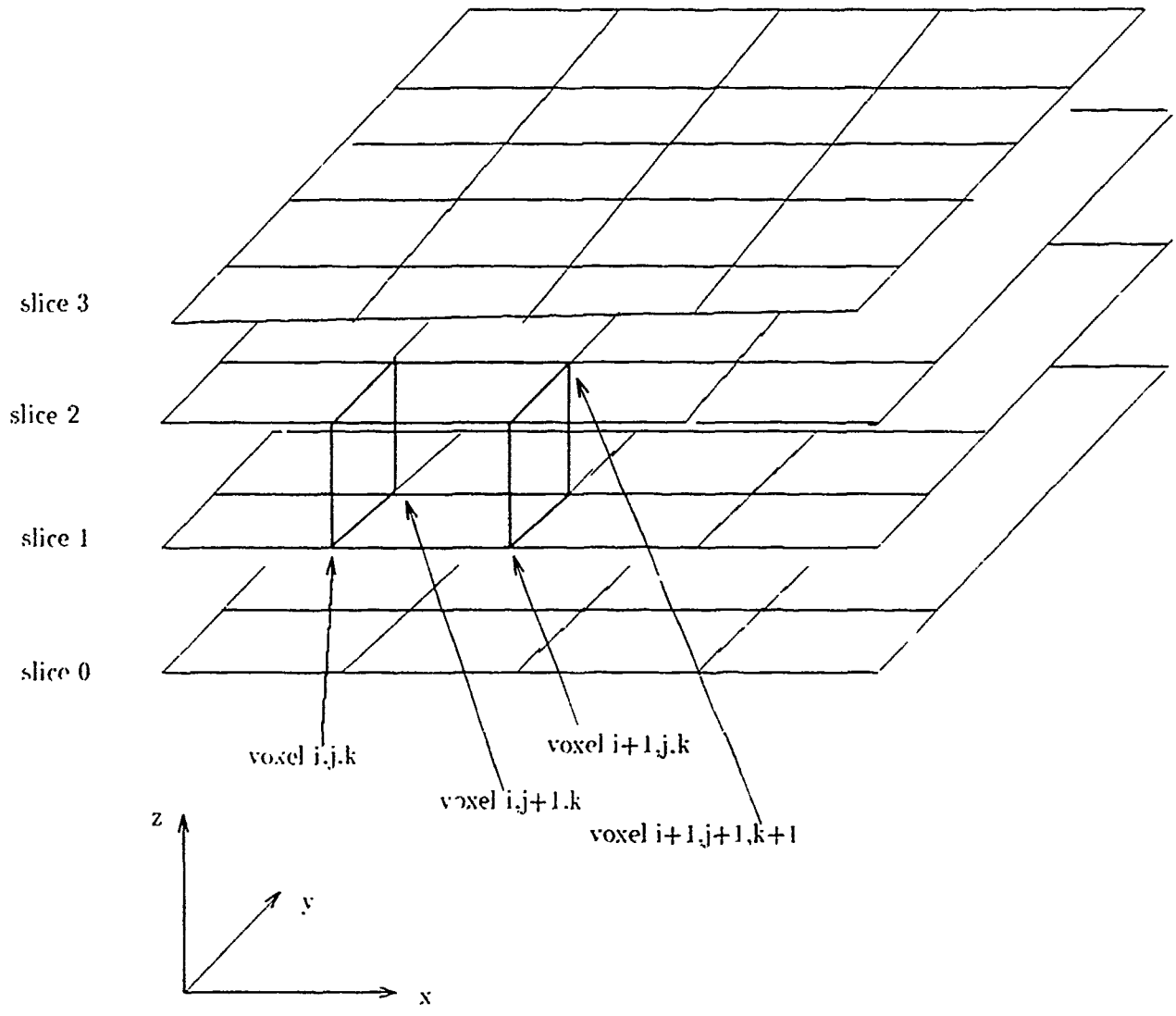


Figure B.1. Computational cell (cube) marching between data slices

First, I discuss the implementation of step 4 in the marching cubes algorithm.

B.4 Precalculated Table

Creating the pre-calculated table mentioned in step 4 is actually the first step that must be done. This table contains 256 entries - one for each of the possible cell vertex classifications. Each entry also has associated with it a list of vertices that map to a unique case number (that also accompanies each entry).

To create the table, I first analyzed all the 256 possible cell vertex classification cases to determine the 15 unique cases and mapped the remainders to the unique cases (figures 2.6, 2.7, and 2.8 depict the unique cases). To simplify this process and obtain accuracy, I reduced human error as much as possible. I used tinker toys to represent a cell, with labels attached to the corners and marked to indicate vertex numbers. I also used a presentation graphics package to output 256 pictures of a numbered cell with a numbered segmented rectangle below to hold the binary value of the case (See figure B.2 (a)). I analyzed each case by marking the appropriate numbered labels for the case, complementing the vertices if necessary, and rotating the cell to correspond to the classification of a unique case. Each entry in the 256 element table contains the order of the cell vertices and the corresponding unique case. The order of vertices corresponds to the order of those specified in figure B.2 (a) (0 1 2 3 4 5 6 7). The ordering is arbitrary but requires consistency.

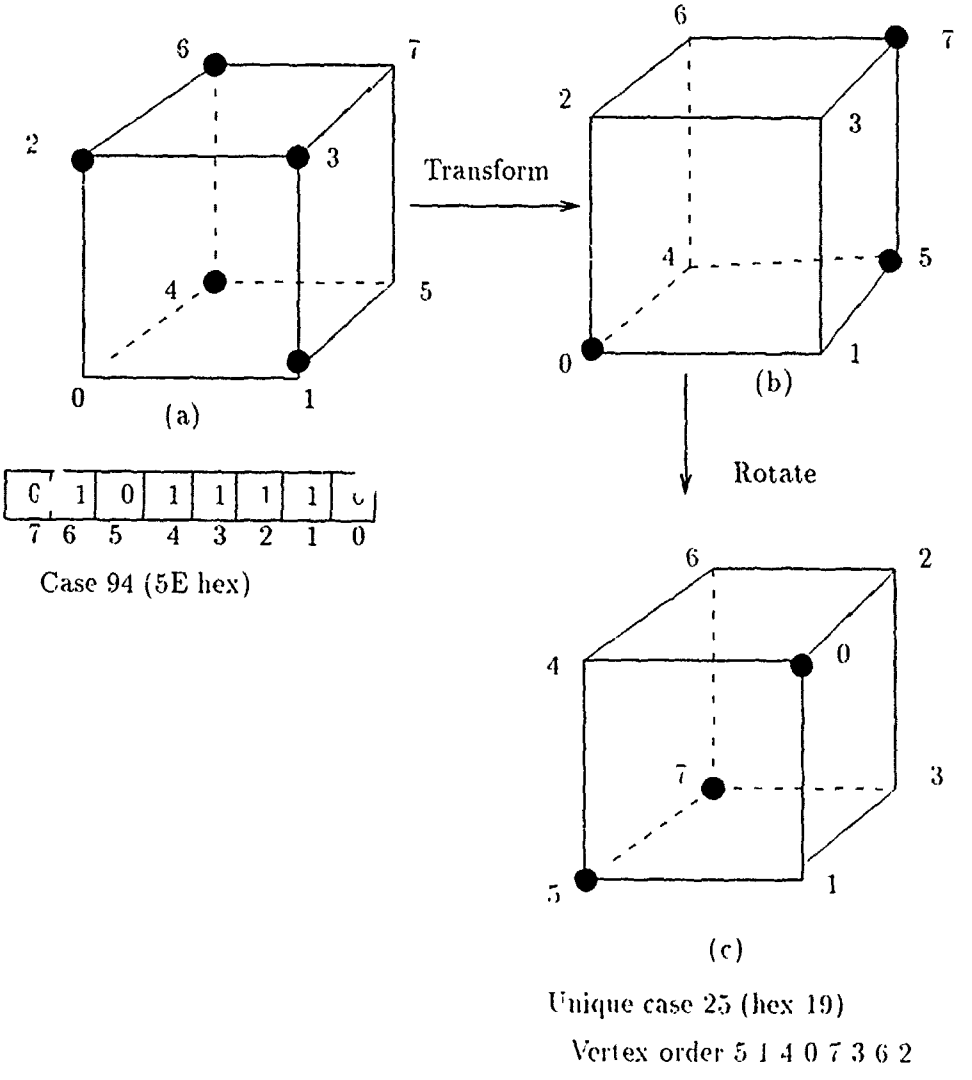


Figure B.2. Example of cell mapping to a unique case

An example of how the table is used will help in understanding its use. Assume we have just marched to the next cell and case 94 (5E hex, 01011110 binary) is encountered. The original orientation is seen in figure B.2 (a). In a case such as this, Lorensen and Cline point out that "Complementary cases, where vertices greater than the surface value are interchanged with those less than the value, are equivalant" (34:165). Therefore, figure B.2 (a) is transformed to figure B.2 (b), which when rotated, matches with the same vertex classification as unique case number 25 (see figure B.2 (c)). The major table entry for case 94 contains 25 for the unique case index and also contains the cell vertex ordering 5 1 4 0 7 3 6 2. Another table called the translation lookup table is used to map case 25 into triangles based upon previously computed interpolation points along the cell edges, and uses the ordering in the table in place of the normal ordering 0 1 2 3 4 5 6 7 (which is only used in unique cases).

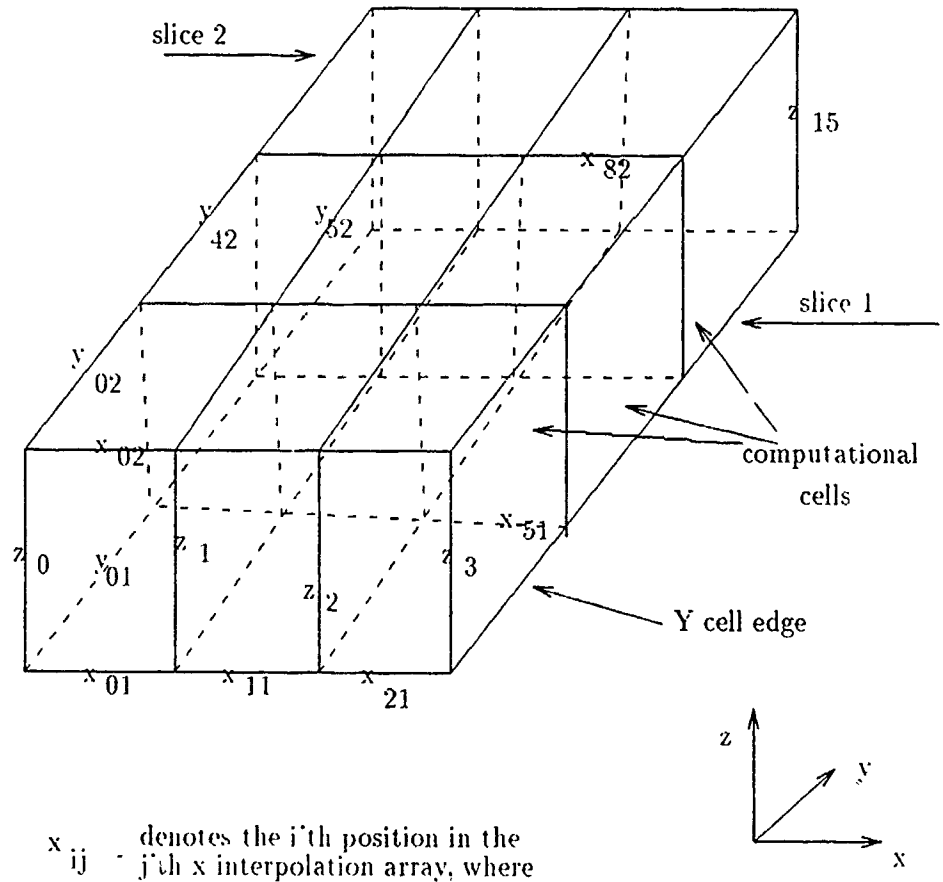
Step 5 is another place where pre-processing before marching can occur to speed up the algorithm.

B.5 Cell Edge Interpolation

Cell edge interpolation is the process that determines where the iso-surface intersects a cell edge. The cell edge must have one voxel value greater than the iso-surface and the other less than the iso-surface for cell edge interpolation to occur

One area where marching cubes implementations can differ is when cell edge interpolation occurs (step 5). My code interpolates in a pre-marching step. That is, every time a new scan plane (slice) of data is read in, it is immediately processed to find the interpolation points along cell edges where the surface is estimated to cross. This requires a total of seven arrays, each with the dimensions of a slice – two for the original slice data, four for the x and y interpolation values for each of the two planes forming the cells and one array for the z interpolation values. Two x and y interpolation arrays correspond to each of the two slices of data that cells will "march" between. Only one array is needed for interpolation points in the z dimension, because of the geometry of the slices and the orientation chosen. This can be seen in figure B.3. which depicts the correspondence between interpolated points along cell edges and these arrays. The $x_{i,j}$'s represent points interpolated in the x direction where, i corresponds to the ith position in the array, j is the slice number, either 1 or 2 (thus 2 x interpolation arrays). Notice the points only lie along the cell edges in the x direction. The $y_{i,j}$ points represent the same for the y interpolation arrays. The z_i's represent interpolated points in the z direction. If the iso-surface does not intersect a particular edge, that corresponding entry in the array will never be accessed.

The following is the interpolation formula used to find the intersection point along a cell edge. This formula applies only along a single component (x,y, or z) since a cell's edge lies only in one of the three coordinate directions.



x_{ij} - denotes the i'th position in the j'th x interpolation array, where j = 1 or 2 and i ranges from 0 to (xdim * ydim) - 1

y_{ij} - i'th position in j'th y interp array

z_i - i'th position in single z interp array

Figure B.3. Correspondence of interpolation arrays to computational cells matching between two slices

Given:

p1, p2 - the component points to interpolate between.

v1,v2 - the scalar value at points p1 and p2 respectively.

iso - the target value interpolated to.

$$interppoint = ((iso - v2) * (p1 - p2))/(v1 - v2) + p2$$

The translation lookup table is later used during marching to access interpolation points in these arrays. The advantage of this pre-interpolation method is that shared triangle vertices are guaranteed to be the same because only one edge is ever processed, which reduces computation time; whereas, during marching each internal edge is processed twice. However, the disadvantage is the memory required to maintain arrays that contain the interpolated points. Even more overhead is required for this method to implement steps 1 and 6, which calculate the triangle vertex normals.

B.6 Normal Calculations

My code produces the normals during the pre-interpolation step. The following central difference gradient operator is used to estimate the outward direction of the surface at a particular voxel (i,j,k) along the three coordinate axes (34:165) :

$$G_x(i,j,k) = (D(i+1,j,k) - D(i-1,j,k))/\Delta x$$

$$G_y(i,j,k) = (D(i,j+1,k) - D(i,j-1,k))/\Delta y$$

$$G_z(i,j,k) = (D(i,j,k+1) - D(i,j,k-1))/\Delta z$$

D(i,j,k) is the density value at voxel i,j,k and $\Delta x, \Delta y, \Delta z$ are the lengths of the cell edges in the corresponding component. Once the cell normals are estimated, they are interpolated along cell edges to the iso-value using the same interpolation formula used to find the triangle vertices. In the vanilla marching cubes algorithm I handle boundary cases as special cases when determining the vertex normals. That is, voxels along the border of the enclosing rectangular volume are assigned the outward facing normal along the enclosing volume (see figure B.4).

B.7 Fixes to Public Domain Code

The original code I started with was written to be fast (though it was practically unusable because it did not produce normals nor planar polygons) without regard to maintainability or understandability. Therefore, I had to re-engineer a significant portion of it to obtain these software engineering goals. The main problem the code had was that it did not maintain a standard logical order of the original slices when reading them in from the data files. Only the interpolated planes were swapped to maintain order. Planes in the code are the same as arrays corresponding to slices. Planes are swapped to re-use the previously read slice for the next processing loop of cells. Swapping planes is the same as swapping pointers. By swapping

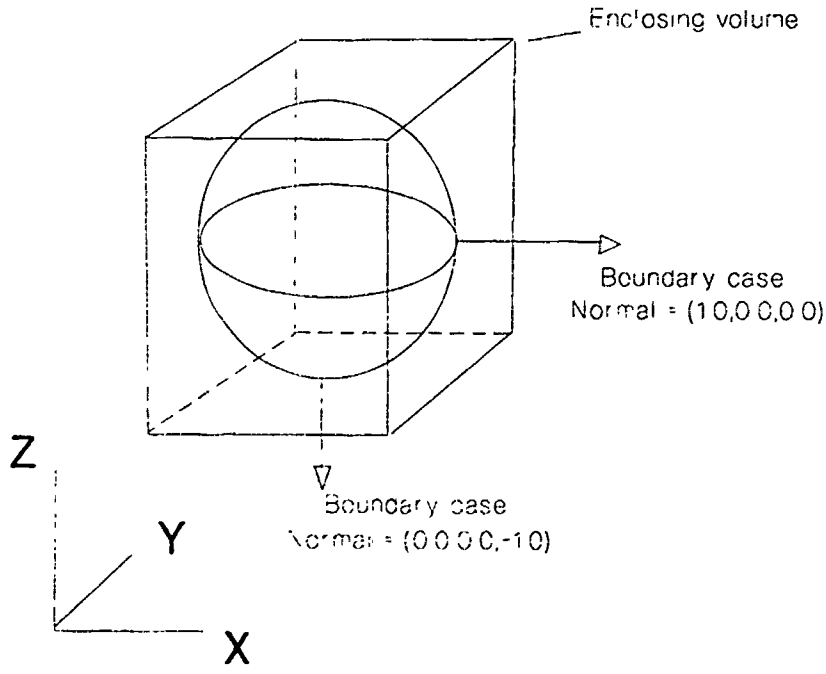


Figure B.1. Vanilla marching cubes example boundary cases in surface normal estimation

all planes after each slice is processed. I cut the size of the translation table in half, but it has to be re-initialized after each slice is processed. However, the code is much easier to understand and modify.

B.8 Marching Cubes Output

The triangles are output in the form of an AFIT geometry file, which expects all the points to be listed first, followed by a list of polygons whose vertices are indicated by referencing line numbers of the above mentioned points. The AFIT geometry file is then used as input to a slightly modified version of AFIT's GPR.

The code for the vanilla marching cubes implementation was written originally in 'C', and consequently it was functionally oriented. It was later converted to C++ so it could interface with the C++, object oriented kriging code.

Appendix A contains data flow diagrams and program description language statements for the vanilla marching cubes implementation.

Appendix C. *Cell Subdivision Implementation*

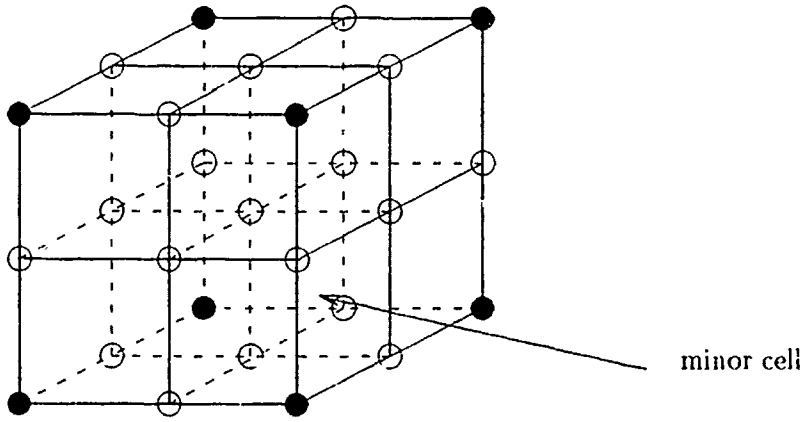
This appendix describes implementation details of the subdivision algorithm in more detail than presented in chapter four of the thesis. First discussed is some terminology to help understand the rest of the appendix. After that, the purpose of cell subdivision is presented. Then the cell subdivision implementation steps are described.

C.1 Terminology

Before proceeding, I present some terminology that eases the following explanations. I consider the initial computational cells in the main loop as major cells (see figure C.1). The newly created cells within a major cell, I term minor cells. Major cells have the original voxels as vertices, so I consider minor cell vertices as minor-voxels. Finally, I consider the arrays within a major sub-divided cell as mini-slices, since logically they represent input data slices.

C.2 Purposes of Cell Subdivision

The primary purposes of cell sub-division are to disambiguate ambiguous cell cases and to derive a better approximation of the iso-surface. An ambiguous cell occurs whenever more than one topology can be chosen for the cell. Cell subdivision disambiguates an ambiguous cell by subdividing it into minor cells. Even though the subdivided ambiguous major cell is no longer ambiguous (because it is no longer dealt



Subdivision factor = 2 in all dimensions

● Original voxels (major cell vertices)

○ Minor voxels (minor cell vertices)

Figure C.1. Subdivided Major Cell

with). cell subdivision does not guarantee that minor cells will not be ambiguous. Wilhelms and Gelder (50) handle ambiguous minor cells by choosing one topology based on facial averages. discussed in chapter two. The cell subdivision method can also be used to increase image fidelity. This is done by forming the surface within minor cells versus the much larger major cells – that is. data resolution is increased by forming minor cells. Increasing the data resolution to improve image fidelity is the same idea behind the dividing cubes algorithm (8). except point primitives are output in the dividing cubes method instead of the triangles output in this cell subdivision method[1]. Subdividing can also be used to increase resolution in only one

[1]The appendix entitled Disambiguation and Enhanced Surface Representation by Cell Subdivision presents an example that helps clarify why cell subdivision can improve image quality

dimension, equivalent to creating cubic voxels in the cuberille data model approaches by creating new logical slices between the original ones. This method is used in the implementation of slice interpolation discussed in chapter four.

C.3 Implementation Steps

This section lists the cell subdivision steps I implemented and describes each in turn. The steps are :

1. Read data into memory.
2. March major cells between slices.
3. Subdivide major cells into minor cells.
4. Estimate minor-voxel values.
5. Apply marching cubes surface extraction within major cells to form surface.

Step 1 Read Slices

Step 1 is basically the same step as in the vanilla marching cubes (vmc) implementation described in the previous appendix. Four slices are needed at a time to calculate the gradient. The central difference gradient operator requires 64 voxel values surrounding a major cell for the calculation (see figure 4.3)[2]. This step differs from the vmc implementation in the handling of border cases. The vmc code deals with border cases (such as the first and last data slices and edges of data slices) by

[2]The process of gradient calculation within minor cells is discussed in a later section

approximating the surface normal at these major cell vertices with a method other than the central difference gradient operator. This is because the 64 surrounding voxel values do not exist in these cases. Instead of the gradient operator in these cases, the normals are approximated by the vector normal to the surrounding parallelepiped shaped volume (see figure B.4.

Since two of the estimation techniques (tricubic interpolation and kriging) used in the cell subdivision process need 64 surrounding values at all times, I handled the border cases differently in the cell subdivision implementation. First, I always assume there are 64 surrounding voxel values. To do this, I ignore the data on the edges of the 2D data slices used as input. This is not a problem because in most medical image data slices several rows and columns of edge values do not contribute to the meaningful portion of the data. Because of this assumption, I insure the artificial volumes I create are centered within the volume, with at least one array position in the x and y dimensions as a buffer zone. Since I cannot ignore the first and last data slices, I create two dummy data slices to replace them. The values for these dummy slices are copied from the slice they are imitating.

Step 2 March

Step 2 is the same marching that occurs in the vmc implementation. Computational cells are marched between two data slices (see figure 4.1). In cell subdivision, these cells are not polygonized, but are subdivided. This subdivision is described in the next step.

Step 3 - Cell subdivision

The purpose of cell subdivision was described above – here the implementation is discussed.

I process the subdivided major cell by employing another modified version of my vmc implementation to "march" within each subdivided major cell. Since I use a vmc implementation, data slices are assumed to be read into memory; therefore, I simulate reading data slices into memory. Since I never use more than four data slices at a time, I use four arrays to hold the mini-slice values and points. In the vmc implementation, data slices already have values and points associated with them when read into arrays in memory. However, both values and points must be calculated for mini-slice arrays. The points are calculated from subdivision factors specified prior to execution. Three subdivision factors along each of the three major axes are set (e.g., fx = 3, fy = 5, fz = 2 means subdivide the cell in the x direction into three parts, in the y direction into five parts,etc.). Figure C.1 depicts a major cell subdivided into eight minor cells where the subdivision factor is two in each direction. A subdivision factor of two in each direction requires the calculation of one minor-voxel point at the midpoint of each major cell edge, one minor-voxel point in the center of each major cell face, and one minor-voxel point in the very center of the major cell. Figure C.2 depicts a major cell subdivided into 5 parts in all three directions. The calculation for dividing a major cell edge into 5 parts is

$$pl.x = major_{x1} + (major_{x2} - major_{x1})/5$$

$$p2.x = major_{x1} + 2 * (major_{x2} - major_{x1})/5$$

$$p3.x = major_{x1} + 3 * (major_{x2} - major_{x1})/5$$

$$p4.x = major_{x1} + 4 * (major_{x2} - major_{x1})/5$$

The y and z components are calculated in the same manner.

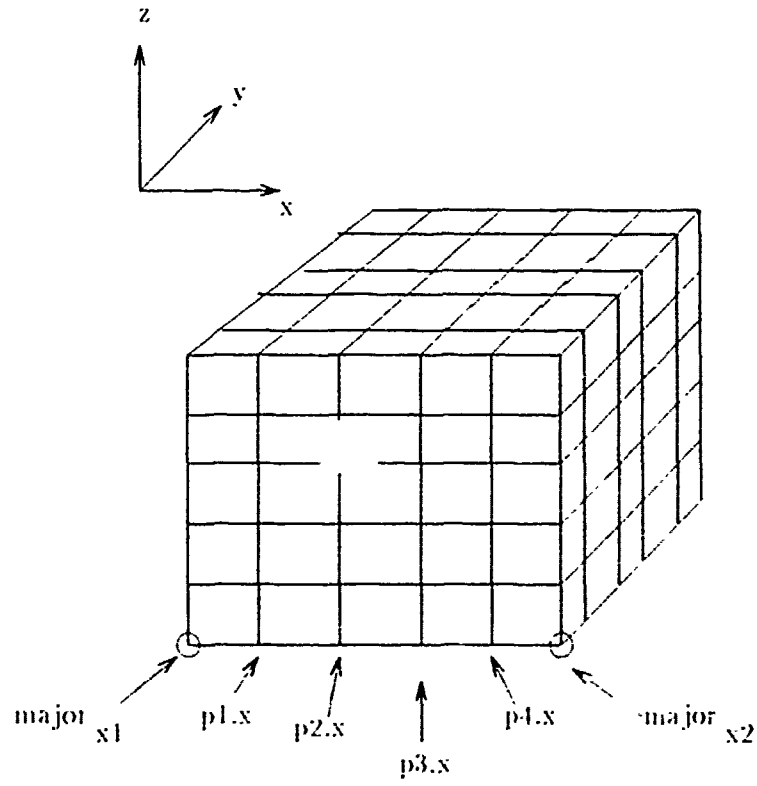


Figure C.2. Computing subdivision points. Subdivision factor = 5 in all three directions.

Once a minor-voxel point is determined, a scalar value is assigned to the point.

Step 4. Estimate scalar values

In order for the modified vmc implementation to process mini-slices, scalar values must be estimated at the minor cell vertices. I implemented three functions to estimate values at minor cell vertices. These are trilinear interpolation, tricubic interpolation and kriging. Trilinear interpolation is explained in a separate appendix. Tricubic interpolation is explained in chapter four. Kriging theory is presented in chapter three and the implementation details are in chapter four. Once the minor cell vertices are estimated, the surface can be formed.

Step 5. Apply marching cubes surface extraction within major cells

I implemented two modified versions of the vmc implementation described in the previous appendix to accomplish cell subdivision. The first vmc implementation is used in the outermost loop to read the actual data slices into memory. There are two primary loops in the system. Within this outermost loop, major cells are formed. The only tasks the first vmc implementation does is read data into memory and form major cells. The surface is actually formed by the second vmc implementation, which processes major cells.

The second vmc implementation treats major cells as sub-volumes, extracting sub-surfaces from them. Thus the second primary loop "marches" within major cells, forming portions of the iso-surface. That is, for each major cell within a volume of 3D data, the second vmc implementation extracts a sub-surface from each sub-volume (major cell). The major task here is to insure there is a continuous surface extracted

not only between minor cells but also between sub-volumes. A continuous surface means the triangle vertices and normals are the same at shared locations. Fortunately the vmc implementation described in the previous appendix insures surface continuity between minor cells by a pre-marching interpolation step. However, the task of insuring surface continuity between sub-volumes is not as straightforward.

The challenge of dealing with inter sub-volume surface continuity determines how minor cell vertex normals are approximated on the boundary of major cells. Recall from the vmc algorithm that surface normals are approximated at cell vertices by a central difference gradient operator. I use this operator to approximate normals at all minor cell vertices completely contained within a major cell. However, the minor cell vertices on the borders of major cells are handled specially to insure inter sub-volume surface continuity. First, those minor cell vertices that are the same as the major cell vertices are assigned the same normal value as the major cell vertices. This is possible because pre-marching interpolation of both points and normals is performed in the outermost loop. Then, prior to marching within a major cell, I calculate cell face normal averages and cell edge normal averages to use on the other boundary cases. These normal averages shared between sub-volumes insures inter surface sub-volume continuity.

After the points, values, and normals are estimated for the minor cell vertices, I then interpolate points and normals in a pre-marching step (the same as in the vmc implementation described in the previous chapter) to determine the surface-minor

cell intersections. Next, I "march" minor cells between mini-slices and output triangle vertices and normals to an Air Force Institute of Technology (AFIT) geometry file. To render the surface, I call a modified version of the AFIT General Purpose Renderer (GPR) to do Phong illumination and Phong shading. Another appendix describes the modifications to the AFIT GPR.

Appendix D. *Disambiguation and Enhanced Surface Representation by Cell Subdivision*

This appendix describes how cell subdivision can disambiguate ambiguous cells and how it can enhance the surface representation. First, some background information is presented, then an example is presented that helps demonstrate the purpose of this appendix.

D.1 Background

I, as well as Wilhelms and Gelder (50) demonstrate that subdividing cells reduces ambiguity significantly, and depending on the estimation function used to estimate intra-cell scalar values, can cause a smoother representation of the surface generated by cell interpolation. Using a subdivision factor of 5 in each dimension, I explore both trilinear and tricubic estimation functions in artificial volumes. The trilinear function generates a better surface fit than the vanilla marching cubes, but is still far from the desired surface. Tricubic estimation even performs better in these artificial volumes. Tricubic estimation causes the surface extraction to generate a closer representation of the actual surface than the trilinear does. The authors cited above claim the tricubic is better at estimating points within the cell because it uses a larger neighborhood of points without assuming linearity. However, this assumption may not be valid for data with sharp contrasts within a small neighborhood of voxels.

A larger neighborhood may in fact cause errors in data with sharp contrasts. Kriging estimation can also use a larger neighborhood of control points, The promising nature of kriging is that it guarantees the "best" linear estimator and the neighborhood size can be modified as well as a number of other parameters.

Subdividing ambiguous cells does not guarantee that ambiguous cells will be removed. The minor cells created from the subdivision process may be ambiguous. Wilhelms and Gelder (50) apply the facial averaging technique, described in chapter two, to disambiguate minor cells. That is, if a face is ambiguous, the average value obtained by averaging the four face vertices is tested against the iso-value. If the average is greater than the iso-value, the 1-vertices are connected, else the 0-vertices are connected.

In all the artificial data sets I create, ambiguity is completely removed without the need for disambiguating minor cells. However, this does not occur in the medical data sets I tested. In the artificial data sets, the surface generated by cell interpolation appears smoother the higher the subdivision factors and depending on the estimation function. The next section explores how this smoother surface representation can occur.

D.2 Example

The polygonization in the marching cubes algorithm or any cell interpolation algorithm is arbitrary. It is a guess at how the surface should pass through the

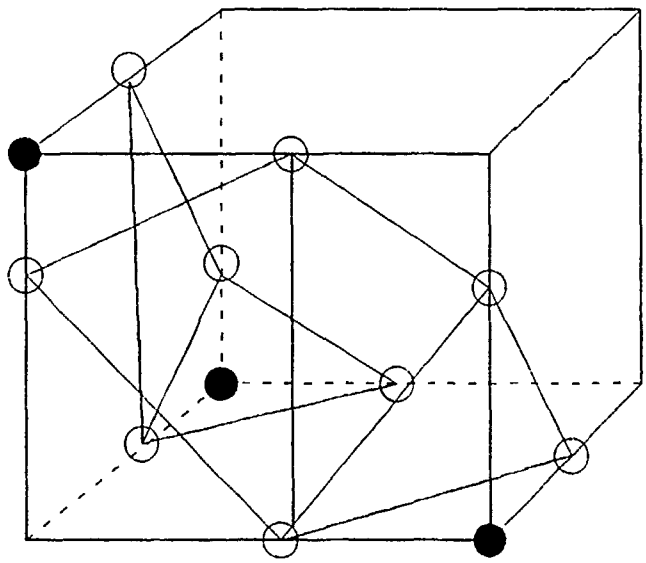


Figure D.1. Alternate polygonization for case 6

cell. For example, another triangulation of unique case 6 of figure 2.7 is depicted in figure D.1.

Note that only the ambiguous cases of the unique case figures in chapter two are truly ambiguous in the sense that alternate polygonizations can be performed. For example, the polygonization of unique case 5 (a non-ambiguous case) is fairly obvious, even though it is remotely possible further subdivision and estimation could uncover a different topology[1].

To understand how cell subdivision and estimation can disambiguate ambiguous cells, consider a major ambiguous cell in which a possible erroneous topology is generated (major cell refers to an un-subdivided cell, a minor cell a subdivided cell).

[1]Topology refers to the polygonization of a cell that represents a portion of the surface topology

Without subdivision, erroneous topology is a gross error, which not only causes inaccuracies in the final image, but can also create "holes" as discussed in chapter two. Sub-dividing this cell will reduce this one large erroneous topology into smaller cells where most will have non-ambiguous cells, depending on how well the estimation function estimates the surface in the cell. Case 14 (see figure D.2) is a particularly good example of a rare case, even if no ambiguity results. The reason why it is rare is because it represents a very complicated portion of the surface topology.

Suppose case 14 is subdivided by a factor of 2 in all three directions. The chances of case 14 appearing within any of the minor cells is even rarer. This is so because of the complex triangulation of case 14. Assume the triangulation of figure D.2 A depicts the surface correctly within that cell. Then subdividing the cell could possibly generate the subdivided major cell depicted in figure D.2 B. In this case the topology remained the same, and case 14 does not show up. In fact, all nonempty minor cells are unique nonambiguous case 1 in this figure. In any subdivision, the 1-vertices of the major cell will remain 1-vertices in the corresponding minor cells because their values do not change. However, new 1-vertices may be added on the minor cells. In this figure, the 1-vertices are the same. For simplicity, I will discuss only one edge of the major cell where the surface intersects. This edge is denoted by E figure D.2.

Figure D.2 C depicts possibly different minor cell topologies caused by the subdivision. Note in figure D.2 B that the surface intersects between minor voxels

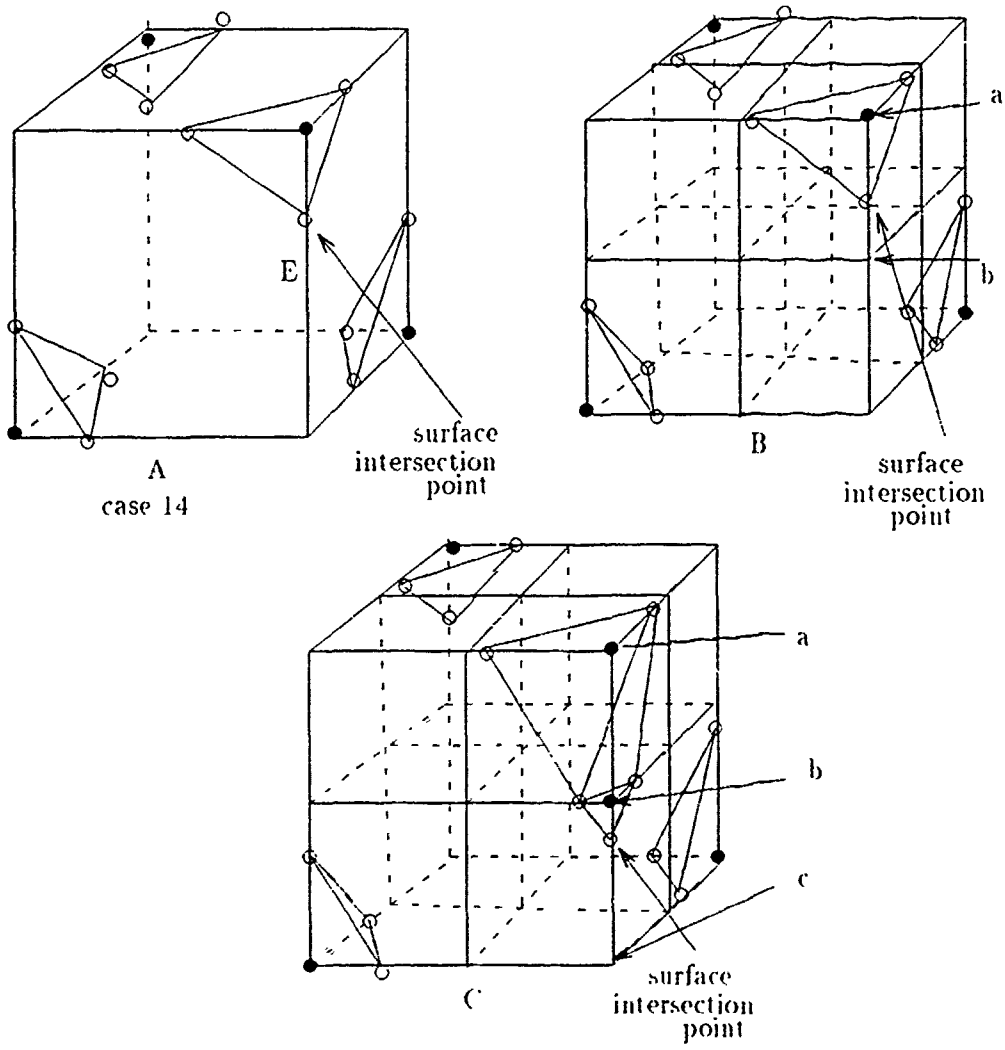


Figure D.2. Example of alternate surface representation caused by cell subdivision

a and b. Figure D.2 C however shows more complicated minor cell vertex classifications, where the surface intersects between minor voxels b and c. In figure D.2 C, the upper right front cell is now unique case 3 instead of 1 and a new nonempty minor cell exists - the lower right front cell. This cell is another unique case 1. This does not imply that figure D.2 A is a wrong topology, it just means that in this case, figure D.2 C captures the surface intersection point on edge E more accurately.

Of course there are many other possible topologies within the minor cells, too numerous to list here. The point of the simple example presented is to show that subdivision can disambiguate and cause the cell interpolation to provide a closer approximation of the actual surface. If minor cell vertex value estimation is accurate, the new surface intersection points should be closer to the true surface boundary, thus generating a triangular mesh that better approximates the actual surface of interest. Subdividing the cell even further should generate even closer surface intersection points. Again, resolution of ambiguous minor cells is not guaranteed, but as stated previously, the few ambiguous minor cells that may result can be dealt with by facial averaging, gradient consistency heuristics, or similar methods.

Appendix E. *Binary Image Format to Utah RLE Format Conversion*

It is often very useful to look at just a single slice of data, especially if the data is from CT or MRI scanning technologies. However, most of the data is in binary, so it must be converted to an image format. I chose the Utah RLE format because I can view an RLE image from any of the different types of workstations we have at the Institute. My code assumes binary files as input, but can be easily modified to read from ASCII files. The code is in the directory thesis/src/makerle

The majority of image files are stored as 1 byte per value, with values stored in scanline order from bottom to top. Therefore the double for loops used to read and write from/to these files increases fastest in x.

Some differences in file formats were discovered during this effort. The Chapel Hill data has 2 bytes per value. In this case, it is important to determine if the bytes need swapping because of architecture differences (little endian/big endian). Also, some MRI image data derived from an unknown source was stored as sun raster files. To determine if the image format is sun raster, try the sun call (on a Sun console) *screenload* <image-file> or the Unix command *file*.

The Utah RLE toolkit has routines that do the same task (graytorle and rawtorle); however, the code I developed allowed me to perform other operations on the data. For example, I modified it to compute the histogram of the data. The major

benefit of the code is that it has the basic format for reading binary data, which is necessary in other areas, such as the actual rendering program.

Appendix F. *Changes to the Air Force Institute of Technology's General Purpose Renderer*

AFIT's GPR is an object oriented rendering system written in C++. It has several hidden surface removal, reflection model, and shading model implementations. This plethora of options consequently makes the executable quite large. Since memory is a scarce resource when dealing with volumes, I decided to minimize GPR's memory usage. I did this by removing unnecessary portions such as code handling texture mapping, vertex colored polygons, Bezier patches, scan line z-buffering, etc. I retained the 'Z' buffer and 'A' buffer hidden surface removal implementations, all the reflection models, and both flat and Phong shading.

Although this reduced the executable size by over a half, GPR still required too much memory for a single geometry file output from my marching cubes implementation. I attempted to fix this problem by creating a list of geometry files, since GPR is capable of reading multiple files. The code is supposed to free memory after processing each geometry file; however, the deallocation did not appear to work correctly.

GPR allocates many arrays. It is possible the GNU C++ array deallocation does not work properly. When array deallocation is attempted according to Schildt (38:337), the g++ warning message "array size expression for delete ignored" results

(e.g. delete [pcount] pnumverts). Schildt indicates the importance of this operation (ignored by g++) :

> One reason that you need to specify the number of elements in the array to the **delete** operator is so that the proper number of destructor functions can be called (that is, one for each object in the array).

However, after I investigated memory allocation and deallocation by creating test cases using the same data structures used by GPR, I discovered that the array size had no affect on deallocation. The main factor appears to be the order and the sizes of the memory blocks allocated. I found that if a smaller block of memory is allocated first and then freed, a larger block cannot use the just freed space because it is not large enough (both malloc(),free(), and new,delete appeared to operate the same). Therefore, for the best use of Unix memory management, large blocks should be allocated first.

This knowledge still did not solve the problem of GPR using too much memory, because GPR allocates many blocks of varying size, based on the polygon count, the vertex count and the number of vertices per polygon. To keep from altering GPR significantly to order memory allocations correctly, I chose to implement a very simple memory management scheme.

The average memory required for processing geometry files output from marching cubes is between three and five megabytes per file. If GPR successfully makes this amount available in heap space after every file is processed, then no memory

problems result. Therefore, I decided to create my own heap by simply allocating one block of memory large enough to handle the largest file in the list of files output from marching cubes. This size varies based on the isovalue(s) selected and the input data resolution; however, I found the maximum needed never exceeds 10 megabytes.

The memory block is allocated once at the beginning of GPR in main() by

memptr = (void *) malloc(MEMORY);

where MEMORY is the pre-defined size.

Since memptr is type casted as a void *, portions of this block can be cast to any type. An offset into this memory block is maintained for assigning new memory. Before each geometry file is processed, this offset is re-set to zero. Thus, no delete or free operations are necessary and the same memory is re-used over and over.

Appendix G. *Creating Artificial Volumes*

This appendix discusses the methods that create an artificial volume. An artificial volume (term taken fr n Tiede (43)) in this work is one in which scalar values are artificially entered at node points in a 3D array to represent some object surface, versus a volume containing voxel values which are generated by a scanning technology such as MRI or CT or by a scientific simulation. The primary artificial volumes I created for this work contain surfaces depicting three-dimensional mathematical functions such as a sphere,an ellipsoid. or a paraboloid. In both methods I implemented, the surface is always centered within the first octant (positive x,y.z) by subracting the center point (h.k,l) from the points (x.y.z) in the math equation.

The initial method I developed is very straightforward, but does not allow setting a surface threshold other than 0. I accomplished this by assigning to each voxel the value returned by evaluating the math function at the voxel 3D point. For example, a voxel value at mesh point (x.y.z) for a sphere is determined by :

$$voxelValue(x,y,z) = value(x,y,z) = (x-h)^2 + (y-k)^2 + (z-l)^2 - r^2$$

where r is the radius and (h.k.l) the center point.

Functions defining a surface return positive values on one side and negative values on the other. where surface points are evaluated at zero. Since marching cubes interpolates triangle vertices to the scalar value between voxels, this method generates a volume in the correct format for marching cubes to read. This is so

because a surface defined by a math function will rarely if ever intersect an artificial volume voxel. Of course, the larger the volume, the better chances this will occur.

Lt Col Phil Amburn extended the method just discussed to allow any scalar value to represent the surface. Since the value returned from a math function evaluated at a mesh point denotes the distance from the surface in a positive or negative direction, this distance is used to taper off a value from the scalar value chosen. Also, to get the output in marching cubes formula, I modified the algorithm to taper off towards the negative if the point is on the negative side of the surface and towards the positive if the mesh point is on the positive side. The formula for this method to determine the value at a mesh point is :

$$dist = fabs(value(x,y,z));$$

if $(value(x,y,z) > 0.0))$ /* positive */

$$voxelValue(x,y,z) = (1 - \frac{dist}{MAXDIST}) * SCALAR$$

$$+(\frac{dist}{MAXDIST}) * (SCALAR + OUTSIDE)$$

else /* negative */

$$voxelValue(x,y,z) = (1 - \frac{dist}{MAXDIST}) * SCALAR$$

$$+(\frac{dist}{MAXDIST}) * (SCALAR - OUTSIDE)$$

I used AFIT's viewit program on the Silicon Graphics 3100 series workstations to view the artificial volumes. I also included a command line option to draw a box around the volume perimeter.

Appendix H. *Trilinear Interpolation*

This method assumes scalar values vary linearly along component directions between voxels (45:11-14). It interpolates in each of the three dimensions, see figure H.1, where f(Oi) and f(ni) represents the scalar value at original point i and new (or intermediate) point i, respectively. The goal is to estimate f(n6) given f(O0) - f(O7).

The method computes ratios of distances and scalar value differences successively until the center value is determined. That is , $\frac{f(n0)-f(O0)}{d2} = \frac{f(O1)-f(O0)}{d1}$

$f(n0) = \frac{d2}{d1}(f(O1) - f(O0) + f(O0)$

Then, f(n1) is found by interpolating between f(O2) and f(O3) in the same manner. f(n5) is found by interpolating between the intermediate values f(n1) and f(n3). The final value f(n6) is determined by interpolating between f(n4) and f(n5).

There is one significant problem with this method that reduces accuracy of estimates in a three-dimensional data set. Only the eight surrounding voxels are analyzed to estimate a new value at a particular point within the cell, when in fact these eight samples may not provide sufficient information to infer the variability of the data. More importantly however, the data may not vary linearly along major cell edges.

A problem of less significance is the direction in which to interpolate initially is just an arbitrary assumption (though only three choices exist). This is important

because f(n4) and f(n5) are based on the first set of intermediate interpolations in this assumed direction (e.g., the Y direction could be assumed initially – then f(n0) would be determined by interpolating between points O1 and O5). This is a potential cause of inaccuracy because ,for example, values may not vary the same between voxels O0 and O1 and between O1 and O5.

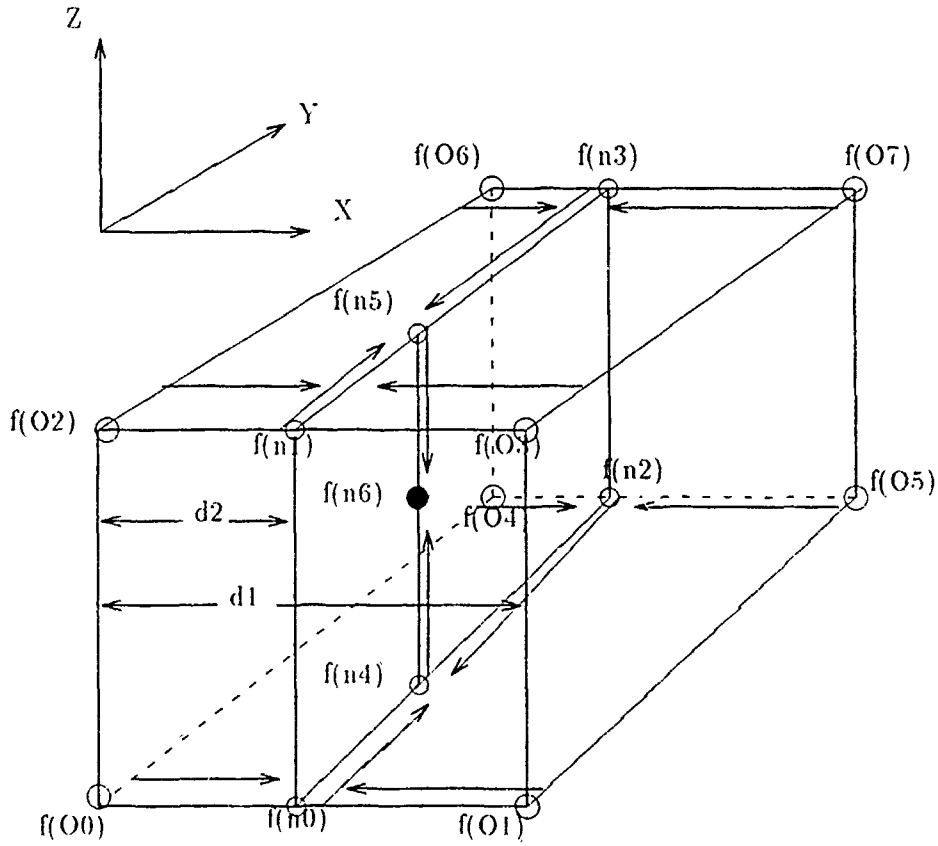


Figure II.1. Trilinear interpolation

Bibliography

1. Artzy, Ehud, et al. "The Theory, Design, Implementation and Evaluation of a Three-Dimensional Surface Detection Algorithm," *Computer Graphics and Image Processing*, *15*:1–24 (1981).
2. Artzy, Ehud, Gideon Frieder and Gabor T. Herman. "The Theory, Design, Implementation and Evaluation of a Three-Dimensional Surface Detection Algorithm," *Computer Graphics*, *14(3)*:2–9 (July 1980).
3. Brodkin, Chris. *The Application of Kriging for Controlled Minimization of Large Data Sets*. Draft MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.
4. Chen, L., et al. "Surface Shading in the Cuberille Environment," *IEEE Computer Graphics and Applications*, *5(12)*:33–43 (December 1985).
5. Christiansen, H. N. and T. W. Sederberg. "Conversion of Complex Contour Line Definitions into Polygonal Element Mosaics," *Computer Graphics*, *12(3)*:187–192 (1978).
6. Clark, Isobel. *Practical Geostatistics*. London: Applied Science Publishers Ltd, 1979.
7. Cline, H.E. et al. "3D Reconstruction of the Brain from Magnetic Resonance Images Using a Connectivity Algorithm," *Magnetic Resonance Imaging*, *5(5)*:345–352 (1987).
8. Cline, Harvey E., William E. Lorensen et al. "Two Algorithms for the Three-Dimensional Reconstruction of Tomograms," *Medical Physics*, *15(3)*:320–327 (May/June 1988).
9. Cressie, Noel. "Kriging Nonstationary Data," *Journal of the American Statistical Association*, *81*(395):625–631 (September 1986).
10. Cressie, Noel. "Spatial Prediction and Ordinary Kriging," *Mathematical Geology*, *20*(4):405–421 (1988).
11. Cressie, Noel. "Geostatistics," *The American Statistician*, *43*(4):197–202 (November 1989).
12. Cressie, Noel. "The Origins of Kriging," *Mathematical Geology*, *22*(3):239–52 (1990).
13. David, Michael. *Geostatistical Ore Reserve Estimation*. New York: Elsevier Scientific Publishing Company, 1977.
14. Davis, John C. *Statistics and Data Analysis in Geology* (Second Edition). New York: John Wesley & Sons, 1986.

15. Delfiner, P. and J. P. Delhomme. "Optimum Interpolation by Kriging." *Display and Analysis of Spatial Data* edited by John C. Davis and Michael J. McCullagh. 96–114, New York: John Wiley & Sons, 1975.

16. Drebin, Robert A., et al. "Volume Rendering," *Computer Graphics, 22(4)*:65–74 (August 1988).

17. Dubrule, Olivier. "Comparing Splines and Kriging," *Computers & Geosciences. 10*(2-3):327–38 (1984).

18. Durst, Martin J. "Letters: Additional Reference to Marching Cubes," *Computer Graphics, 22(2)*:72–73 (April 1988).

19. Farrell, Edward J. and Rosario A. Zappulla. "Three-Dimensional Data Visualization and Biomedical Applications," *CRC Critical Reviews in Biomedical Engineering, 16(4)*:323–363 (1989).

20. Foley, James D., Andries van Dam Steven K. Feiner and John F. Hughes. *Computer Graphics: Principles and Practice* (second Edition). Addison-Wesley Publishing Company, 1990.

21. Fox, John. *Linear Statistical Models and Related Methods With Applications to Social Research*. John Wiley & Sons, 1984.

22. Frieder, Gideon, et al. "Back-to-front Display of Voxel-Based Objects," *IEEE Computer Graphics and Applications, 5(1)*:52–60 (January 1985).

23. Fuchs, Henry, et al. "Optimal Surface Reconstruction from Planar Contours." *Communications of the ACM, 20(10)*:693–702 (October 1977).

24. Ganapathy, S. and T. G. Dennehy. "A New General Triangulation Method for Planar Contours," *Computer Graphics, 16(3)*:69–74 (July 1982).

25. Grant, Michael. *The Application of Kriging in the Statistical Analysis of Anthropometric Data Volume I*. MS thesis, AFIT/GOR/ENY/ENS/90M-8. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, May 1990 (AD-A220 613).

26. Herman, Gabor T., et al. "Computer Techniques for the Representation of Three-dimensional Data on a Two-dimensional Display," *SPIE, 367*:3–11 (1982).

27. Herman, Gabor T. "A Survey of 3D Medical Imaging Technologies," *IEEE Engineering in Medicine and Biology, 9(4)*:15–17 (December 1990).

28. Herman, Gabor T. and Jayaram K. Udupa. "Display of 3-D Digital Images: Computational Foundations and Medical Applications," *IEEE Computer Graphics and Applications*, 39–46 (August 1983).

29. Journel, Andre G. *Fundamentals of Geostatistics in Five Lessons*. Washington, D. C.: American Geophysical Union, 1989.

30. Keppel, E. "Approximating Complex Surfaces by Triangulation of Contour Lines," *IBM Journal of Research and Development*, *19(1)*:2–11 (January 1975).

31. Kerbs, Lynda. "GEO-Statistics: The Variogram." *COGS Computer Contributions*, *12*(2):54–59 (August 1986).

32. Laur, David and Pat Hanrahan. "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics*, *25(4)*:285–288 (July 1991).

33. Levoy, Marc. "Volume Rendering and Display of Surfaces from Volume Data." *IEEE Computer Graphics and Applications*, 29–37 (May 1988).

34. Lorensen, William E. and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm." *Computer Graphics*, *21(4)*:163–169 (July 1987).

35. Matheron, G. "Principles of Geostatistics." *Economic Geology*, *58*:1246–66 (1963).

36. McCormick, Bruce H., et al. "Visualization in Scientific Computing," *Computer Graphics*, *21(6)*:1–7, A-1 to C-18 (November 1987).

37. Mendenhall, William, et al. *Mathematical Statistics with Applications*. PWS-KENT Publishing Company, 1990.

38. Schildt, Herbert. *C++: the Complete Reference*. McGraw-Hill, 1991.

39. Stytz, M. R. and O. Frieder. "Three-Dimensional Medical Imaging Modalities: An Overview," *Critical Reviews in Biomedical Engineering*, *18 Issue 1*:27–54 (27-54 1990).

40. Stytz, Martin R. and Ophir Frieder. "Computer Systems for Three-Dimensional Diagnostic Imaging: An Examination of the State of the Art." *Critical Reviews in Biomedical Engineering*, *19(1)*:1–45 (1991).

41. Stytz, Martin Robert. *Three-Dimensional Medical Image Analysis Using Local Dynamic Algorithm Selection On a Multiple-Instruction, Multiple-Data Architecture*. PhD dissertation, PHD. Computer Science and Engineering at the University of Michigan, 1989.

42. Sunguroff, Alexander and Donald Greenberg. "Computer Generated Images for Medical Applications." *Computer Graphics*, *12(3)*:196–202 (1978).

43. Tiede, Ulf, et al. "Investigation of Medical 3D-Rendering Algorithms." *IEEE Computer Graphics and Applications*, 52–62 (May 1990).

44. Udupa, Jayaram K. "Interactive Segmentation and Boundary Surface Formation for 3-D Digital Images." *Computer Graphics and Image Processing*, *18(3)*:213–235 (March 1982).

45. Udupa, Jayaram K. and Gabor T. Herman, editors. *3D Imaging in Medicine*. Boston: CRC Press, 1991.

46. Upson, Craig and Michael Keeler. "V-BUFFER: Visible Volume Rendering." *Computer Graphics, 22(4)*:59–64 (August 1988).

47. Upson, Craig, et al. "The Application Visualization System: A Computational Environment for Scientific Visualization." *IEEE Computer Graphics and Applications*, 30–41 (July 1989).

48. Watson, G. S. "Smoothing and Interpolation by Kriging and with Splines." *Mathematical Geology, 16*(6):601–15 (1984).

49. Watt, Alan. *Fundamentals of Three-Dimensional Computer Graphics*. Addison-Wesley Publishing Company, 1989.

50. Wilhelms, Jane and Allen Van Gelder. *Topological Considerations in Isosurface Generation*. Technical Report. University of California, Santa Cruz, CA, April 1990.

51. Wilhelms, Jane and Allen Van Gelder. "A Coherent Projection Approach for Direct Volume Rendering," *Computer Graphics, 25(4)*:275–284 (July 1991).

52. Wyvill, G., C. McPheeters and B. Wyvill. "Data Structures for Soft Objects." *The Visual Computer, 2(4)*:227–234 (August 1986).